# Coalgebraic Completeness Theorems for Effectful Process Algebras

*Todd Schmid*

A dissertation submitted in partial fulfillment
of the requirements for the degree of
**Doctor of Philosophy**
of
**University College London**.

Department of Computer Science
University College London

January 4, 2024

I, Todd Schmid, confirm that the work presented in this thesis is my own. Where information has been derived from other sources, I confirm that this has been indicated in the work.

# Abstract

In 1984, Robin Milner proposed a nonstandard interpretation of regular expressions: as behaviours of nondeterministic processes rather than regular languages. This shift in interpretation drastically reduced the set of sound equivalences between regular expressions, making Arto Salomaa's complete axiomatization of the algebra of regular languages unsound under the new interpretation. In the same paper, Milner adapted Salomaa's axioms to give a sound axiomatization of bisimilarity and asked whether his axiomatization was complete.

Milner's completeness problem was open for nearly 38 years before Clemens Grabmayer published a positive solution in 2022. Milner's problem motivated three decades of research into process algebra, the algebraic study of concurrent processes. Working concurrently with the process algebra community between the 1960s and 1990s, Volodymyr N. Redko, John H. Conway, Dexter Kozen, and their collaborators developed Kleene algebra (KA), an abstraction from the algebra of regular languages and generalization of Salomaa's axiomatization. Their work led to the invention of Kleene algebra with Tests (KAT), which extends Kleene algebra with Boolean control flow for reasoning about imperative programs.

A notable fragment of KAT, guarded Kleene algebra with tests (GKAT), consists of only the programs that can be written using the if-then-else and while-do imperative programming constructs. GKAT is a simple and efficient fragment of KAT for which a sound Salomaa-style axiomatization exists. However, the question of completeness for this axiomatization is still open.

In this thesis, I connect the completeness problems of Milner and GKAT using techniques from universal algebra and coalgebra. In the first part of the thesis, I identify the source of difficulty in both these problems: In GKAT and Milner's regular expressions modulo bisimilarity, some finite systems of formal equations

are unsolvable. I show that every coalgebraic completeness theorem comes from a certain structured class of solvable systems of equations, and vice versa.

In the second part of the thesis, I provide a partial completeness result for GKAT. Specifically, I show that the axioms of GKAT are complete for GKAT expressions of a specific form, called skip-free. The proof is a reduction to the completeness theorem for regular expressions modulo bisimilarity.

In the last part of the thesis, I generalize the completeness problems of Milner and GKAT to a wide class of process algebras with algebraic effects, which I call effectful process algebras. Effectful process algebras capture a number of other process calculi than GKAT and regular expressions modulo bisimilarity, including a few process calculi that mix probability and nondeterminism.

I develop the theory of effectful process algebras in general. I give a uniform construction of both the operational and denotational semantics of effectful process algebras and show that the two semantics coincide, as well as a uniform sound and complete axiomatization of bisimilarity in effectful process algebras. The axiomatization includes a generalization of an axiom scheme used by Smolka et al. (2019) to obtain a completeness theorem for GKAT. My general completeness theorem for effectful process algebras instantiates to the bisimulation variant of this concrete completeness theorem for GKAT, as well as to completeness theorems for the other examples mentioned.

**Funding.** Work on this thesis was partially supported by ERC grant Autoprobe (grant agreement 101002697).

# Impact Statement

A wide variety of programming languages exist, each engineered with its own syntax and for its own specific use cases. In most programming languages, there is no (and can be no) algorithmic procedure for determining precisely when two programs produce all the same results given all the same inputs. This leads computer scientists to develop other kinds of equivalences between programs, relations between pieces of code that preserve program behaviour but may not be able to tell every pair of nonequivalent programs apart. It has also led computer scientists to make the same move that many of the other natural sciences have: To study abstract, idealized versions of the objects in question (in this case, programming languages). This thesis is part of the larger effort of providing general frameworks for developing sound and complete principles for logically deducing behavioural equivalences between abstract programs, to better understand the logical content of these equivalences.

The focus of the thesis is on the *completeness* of certain sound program equivalences, specifically whether they give a full account of *bisimilarity* and *behavioural equivalence* in the contexts where they are appropriate. Famously, for programming languages with restricted expressive capabilities, completeness theorems for bisimilarity are highly nontrivial to prove. Completeness theorems can lead to efficient algorithms for deciding program equivalence through proof-search procedures, as well as justify axiomatic approaches to the semantics of programs. Completeness theorems are highly sought-after results in theoretical computer science.

This thesis consists of two case studies in proving completeness theorems and a general recipe for producing complete axiomatizations of bisimilarity in programming languages featuring an (iterable) algebraic effect. The results build on theoretical work in the area of program semantics, which is greatly influenced by universal algebra and coalgebra, iteration, and probability. Notably, this thesis builds on and

aims to generalize

1. the bisimulation analog of *Kleene algebra* [Mil84], which can be used to model interactive and nonterminating processes,
2. the bisimulation analog of *guarded Kleene algebra with tests* [Smo+20; Sch+21], an abstraction of standard imperative programming in which equivalent interactive control flow structures can be efficiently identified,
3. probabilistic variants and extensions of the previous two [SS00; Róż+23], for reasoning about equivalence of randomized algorithms, and
4. process calculi that mix probability and nondeterminism [MOW03; And99].

A notable application of the second and third points above is to a programming language called NetKAT, which can be used to specify and reason about policies in software-defined packet switching networks [And+14; Fos+15; Fos+16].

# Acknowledgements

Without my supervisors, Alexandra Silva and Jurriaan Rot, this thesis could not have happened. I do not know where I would be without the last four years of their support, their knowledge, their cooking, and especially their patience. It is still hard for me to grasp just how many hours they spent reading and commenting on drafts of papers, job applications, presentation slides, and emails[1]. They were sincerely the best supervisors I could have hoped for, and I cannot thank them enough for everything they have done for me.

I also possibly had the best examiners a PhD student could ask for: Samson Abramsky and Bartek Klin. I cannot thank them enough for reading this thesis with such enthusiasm and in such detail.

I had the pleasure and the privilege of writing papers with some incredible scientists over the last few years: My supervisors Alexandra and Jurriaan, Dexter, Tobias, Fred, Wojtek, Larry, and Victoria. I would like to thank all of you for making science an absolute joy. I eagerly look forward to future collaborations.

The last four years would have been miserable without my academic family, many of whom became close friends. To Wojtek: Thank you for the tremendously positive impact you had on my academic and personal well-being the last two years we have known each other. May we never fall out of touch. To Jana: Thank you for making science fun, for catching my mistakes, for saving my life, for trusting me with your cats, and for giving me a getaway in Holland when I needed it most. To Tobias: Thank you for making me feel like an actual computer scientist, against all odds. To Simon: Thank you for looking out for me at the beginning. I would not be here without you. To the long list of amazing people I have not mentioned explicitly: You know who you are, and you should know that I haven't forgotten you. Thank

[1] I tend to overthink these...

you for the good times, no matter how small.

To Angela, who also spent a fair amount of time reading and commenting on drafts of various important documents and emails: The time we lived in London together was the happiest and most productive I have ever been. Thank you for always being there, for painting me pictures, for not letting me give up, for beating my enemies at card games, for making me laugh when I thought I couldn't, for being an avid completionist, and for teaching me so many basic life skills. You deserve more thanks than I can write in words.

To my brother, Dylan: Without you, I would not have made it through the 8 years leading up to this point as unscathed. You got me through some extremely tough emotional moments, especially in the last two years. Thank you for your support. Thank you for helping me move. And thank you for the the millions of laughs along the way. To my mother and father: Thank you for your seemingly boundless encouragement. You give me a home to miss, to come back to, to be proud of. I sometimes hear you say that I did this all on my own, but that is not true.

Thank you to all the people who made me feel at home in Ithaca during my last year: Sloan, David, and Justin, with whom I spent many lunch hours climbing; Palashi and Meet, possibly the best neighbours; E and the other incredible noise artists from *The Electrozone* and the *Synthica* crew, who gave me an outlet when I desperately needed one; and Karuna, who I believe could make anybody feel special.

My acknowledgements section would, of course, be incomplete without a paragraph consisting mostly of dog names. Thank you, Julia, for trusting me with Mahler. And a sincere thank you to Ira, who runs the charity *All Dogs Matter* in London, and to Wendy and the other ladies at ADM for showing up at our doorstep with Chip, Gloria, Alfie, Elsie, Edgar, Badger, Blaze, Phoebe, Nellie, Terry, Max, the intrepid Finbar, Frankie the sweetest, Rex, Lottie, Murphy, Rosie, Honey, Baxter, another Rosie, Bob, and, yes, even Ricardo.

*Here's to biking in completely the wrong direction*
*and making it home anyway, somehow.*

# Contents

# List of Figures

## Chapter 1

# Introduction

General purpose programming languages are immensely complex, and the body of programming and scripting languages in industry use is ever-growing and developing. In response, theoretical computer science has made the same move that many of the other natural sciences have made: To study, instead, the formal properties of abstract and idealized versions of the naturally occurring objects of interest (in this case, programming languages) [Hoa+87]. Reasoning confidently about software requires a formal understanding of the logical underpinnings of programming languages, and abstract programming languages make these logical underpinnings particularly transparent. The mathematical study of abstract programming languages has resulted in general, language-independent mathematical frameworks for the scientific study of programs, with numerous applications to software verification.

This thesis is part of the larger effort in theoretical computer science to better understand the logical structure of abstract programming languages. Its central contribution is a framework for studying a specific family of abstract programming languages, what I call *effectful process algebras*, that includes a number of languages that have appeared in the literature. Effectful process algebras are abstract programming languages consisting of uninterpreted actions and program branching captured by an algebraic theory.[1] Prototypical examples of effectful process algebras are the bisimulation variant of regular expressions studied by Milner [Mil84] and the propositional while programs studied in *guarded Kleene algebra with tests* [KT08; Smo+20]. To provide the necessary background for effectful process algebra, I discuss these two examples in the next section.

[1] In the context of this thesis, the word *effect* refers to an algebraically presented monad. See the text under *Effectful process algebra* in Section 1.2, and under *Algebraic effects* in Section 4.6, for detail.

## 1.1 Background

One notable example of an abstract programming language is the language of regular expressions. Kleene introduced regular expressions in [Kle56] as a syntax for specifying languages of words recognized by finite deterministic automata. In op. cit., Kleene showed that every language accepted by a DFA is a regular language (this is known as *Kleene's theorem*), and furthermore identified a number of rules for deriving equations between regular expressions that denote the same language. Kleene recognized that formal rules are useful from a technical point of view, as they allow for algebraic reasoning in language equivalence proofs. He left as an open problem the task of giving a complete axiomatization, a set of inference rules from which every language equivalence between regular expressions can be derived as a formal equation.

### Kleene algebra

The first complete axiomatization of language equivalence of regular expressions was given by Salomaa in [Sal66]. Salomaa tailored his inference rules specifically to the regular language model. One of his axioms requires a notion of *empty word property* that does not make sense in other closely related structures, like the relation algebras that appear in software verification [KP00] or the tropical semirings that appear in shortest-path algorithms [HM12].

In a paper from 1964, Redko proved that no finite set of equations can axiomatize the algebra of regular languages [Red64]. Conway proposed a number of related, infinitary sets of axioms sound for both relation algebras and language equivalence in his 1971 book [Con71] (where he coined the phrase *Kleene algebra*). Conway left the discovery of a complete axiomatization as an open problem. A complete infinitary axiomatization was presented by Krob [Kro90]. The complete axiomatization that is nowadays referred to as *Kleene algebra* (KA) is due to Kozen [Koz91]. For a more detailed account of the history of the axiomatization of Kleene algebra, see [FS15].

Kleene algebra has since been extended in many directions, to calculi for reasoning about control flow in imperative programs [KS96; Koz96], randomized (or *genetic*) algorithms [Ros00], programs running synchronously [Wag+19] and concurrently [Kap+18], packet-switched networks [And+14], and many others. The first

extension of KA, for reasoning about control flow, is an influential calculus called *Kleene algebra with tests* (KAT).

KAT was introduced by Kozen and Smith in [KS96; Koz96]. Formally, it is the syntactic and axiomatic extension of KA with a Boolean algebra of tests. The Boolean algebra primitives in KAT allow for the specification of imperative programming constructs like if-then-else conditionals and while loops that are guarded by tests, while the regular expression primitives allow for sequencing, iteration, and nondeterministic control flow.

Like in Kleene algebra, KAT expressions are interpreted as languages of a certain form. Language equivalence of KAT terms is decidable using standard techniques from automata theory. However, due to the inclusion of nondeterminism in KAT, the decision problem for equivalence of KAT terms is PSPACE-complete [CKS96] (as is the case for regular expressions). For contrast, language equivalence of states in a finite deterministic automaton is efficiently decidable [HK73].

KAT can encode abstract imperative programs that cannot be expressed with if-then-else and while loops alone. Restricting KAT to its if-then-else and while loop constructs, i.e. restricting to *guarded*[2] KAT (GKAT) expressions, gives rise to a deterministic fragment of KAT [KT08]. Recently, an axiomatization of language equivalence for GKAT expressions was proposed by Smolka et al. [Smo+20], inspired by Salomaa's axiomatization of language equivalence for regular expressions. The authors show that their axioms are sound for language equivalence in GKAT and that language equivalence is efficiently decidable. However, their completeness proof relies on the presence of an additional axiom scheme, a powerful set of inference rules collectively called the *uniqueness axiom*. The authors leave as an open question whether the uniqueness axiom can be derived from the other axioms—or equivalently, whether the Salomaa-inspired axiomatization is complete for language equivalence in GKAT. At the time of writing, this problem remains open.

In an earlier article [KT08], which introduced GKAT expressions as deterministic KAT expressions, Kozen and Tseng construct an example of a finite deterministic automaton that accepts a language not expressible in GKAT. In other words, GKAT fails to satisfy an analogue of Kleene's theorem. Salomaa's completeness proof for

[2] *Guarded* in the sense that branching is determined by a test.

regular expressions relies on Kleene's theorem, and therefore cannot be adapted to GKAT. This was observed by Smolka et al. in [Smo+20], where the completeness problem for GKAT is first stated.

Interestingly, GKAT is not the only place where a failure of Kleene's theorem presented a barrier to a completeness proof for an abstract programming language. Another example can be found in *process algebra*, the algebraic study of communicating processes [BW90].

**Regular expressions in process algebra**

In the 1980s, Milner proposed an interpretation of regular expressions that is not a model of KA [Mil84]. At the time, process algebra was emerging as its own research field, with the introduction of calculi like Milner's CCS [Mil80], Hoare's CSP [Hoa80], and Bergstra and Klop's ACP [BK82]. In [Mil84], Milner observes that regular expressions can be faithfully translated into the syntax of CCS.

CCS terms denote system behaviours, states in a labelled transition system modulo *bisimilarity*. Bisimilarity originates in modal logic [Ben77] and concurrency theory [Mil80; Par81]. Intuitively, two processes are bisimilar when they are indistinguishable from the perspective of an interacting agent. Bisimilarity is efficiently decidable [HS96], implies trace equivalence in labelled transition systems, and often appears in language equivalence checking algorithms [BP13].

Translating regular expressions to CCS terms and then interpreting them as system behaviours gives an interpretation of regular expressions that differs from the language interpretation of regular expressions in two key ways: First and foremost, there are finite processes that are not bisimilar to any regular expression—the relevant analogue of Kleene's theorem fails! Milner left open the problem of characterizing the processes that are bisimilar to regular expressions. Baeten, Corradini, and Grabmayer published one possible solution to this problem in [BCG07].

Second, there are axioms in Salomaa's axiomatization that are not sound with respect to bisimilarity. Milner was able to adapt Salomaa's axiomatization to give a sound axiomatization of bisimilarity, but left completeness as an open problem. Grabmayer recently published a positive solution to this problem [Gra22], which builds on decades of research [Fok97; FZ94; BCG06; Gra05; BCG07; GF20; Gra21].

Milner recognizes in [Mil84] that the main source of difficulty in his complete-

ness problem appears to be the failure of Kleene's theorem for regular expressions modulo bisimilarity. Grabmayer's solution [Gra22] confirms this observation: The most technically challenging part of his work is its careful characterization of the system behaviours given by regular expressions and the structural analysis of these behaviours that allows the completeness proof to go through.

As the reader might recall, Kleene's theorem also fails in GKAT, and it was conjectured in [Smo+20] that this is the main source of difficulty in GKAT's completeness problem. To understand the relationship between Kleene's theorem and completeness, we turn to a general framework for studying stateful systems: *coalgebra*.

**Coalgebra**

Bisimilarity can be generalized to provide useful notions of program equivalence for system types other than labelled transition systems. It is a central notion in a categorical framework for studying state-based systems called *(universal) coalgebra* [Rut00], in which system types are captured by endofunctors on a category. In coalgebra, a state-based system is a function of the form $X \to BX$, where $X$ is the set of states and $B$ is an endofunctor capturing the system type (the $B$ stands for *behaviour*). The endofunctor $B$ comes with its own notion of bisimilarity, and for many canonical system types the corresponding notion of bisimilarity coincides with the established notion from the literature (see any of [Rut00; Rut98; Jac16] for an extensive catalogue of known instances). This includes the operational models of GKAT programs, so-called GKAT *automata*.

Seeing GKAT and regular expressions modulo bisimilarity through a coalgebraic lens was precisely what enabled the developments of this thesis.

## 1.2 Scope of the Thesis

In this thesis, I clarify the relationship between Kleene's theorem and completeness proofs by placing them in a larger coalgebraic context. I draw concrete connections between the completeness problems of GKAT and regular expressions modulo bisimilarity, and generalize these completeness problems to a much larger class of abstract programming languages. I prove several completeness theorems along the way, including partial completeness results for GKAT and a generic completeness theorem for expressive effectful process calculi generalizing the calculus Milner in-

troduced in [Mil84]. In a nutshell, this thesis studies axiomatization problems for abstract programming languages featuring discrete actions and algebraic effects like nondeterminism and conditional branching. In these languages, a characteristic lack of Kleene theorems make completeness results difficult to prove.

**Completeness and Kleene's theorem**

The connection between completeness and Kleene's theorem can be gleaned from Salomaa's completeness proof. Salomaa's proof depends on being able to solve finite systems of equations in terms of regular expressions, and the class of systems that admit such solutions is closely related to the class of automata that correspond to regular expressions. Kleene's theorem tells us that every finite automaton corresponds to a regular expression, but the standard proof of this fact (using state elimination) is really an argument showing that every finite system admits a solution. Failure of an abstract programming language to satisfy a Kleene theorem is a direct consequence of some systems of equations failing to have solutions in that language. This can be a serious barrier to finding a completeness proof. In Chapter 2, we will see that weaker versions of Kleene's theorem suffice, and furthermore characterize the precise situations in which completeness is obtainable.

**Bisimulation** GKAT

To clarify the connection between the completeness problems of GKAT and regular expressions, I introduce a closely related interpretation of GKAT expressions and its corresponding axiomatization, *bisimulation* GKAT. Bisimulation GKAT is an axiomatization that captures the interpretation of GKAT expressions as behaviours of deterministic processes, states in a GKAT automaton modulo bisimilarity. In Chapter 3, we will see a technique for reducing the completeness problem of GKAT to the completeness of bisimulation GKAT, further motivating the study of bisimulation in the context of GKAT.

**Coalgebraic developments**

In the last part of the thesis, I give a coalgebraic generalization of the completeness problems for bisimulation GKAT and regular expressions modulo bisimilarity. This is obtained by observing that the functors capturing bisimilarity of GKAT programs and bisimilarity of regular expressions are of a common form, namely $M(Var + Act \times (-))$,

where *Var* and *Act* are fixed in the beginning (here, *M* stands for *monad*, terminology that will be explained in Chapter 4). By varying the monad *M* in the system type, we are able to capture both labelled transition systems and the aforementioned GKAT automata. Namely, *M* determines the type of *branching* in each system: Concretely, regular expressions (in process algebra) specify labelled transition systems, which branch nondeterministically (this is captured by setting *M* to be the powerset functor), and GKAT expressions specify automata that branch conditionally with respect to a Boolean guard (this is captured by setting *M* to be the partial maps functor).

**Effectful process algebra**

It turns out that the branching types of GKAT and of regular expressions modulo bisimilarity are determined by sets of equational laws. In regular expressions modulo bisimilarity, branching is specified by a nondeterministic choice operator $+$ that acts like the join operator in a semilattice. In GKAT, program branching is specified using an if-then-else operation with a Boolean guard, which is captured by what we call the *guarded algebra* laws[3]. This is reflected by the type of branching in each case, which is computed as the free algebra satisfying the given set of laws: The free semilattice generated by a given set is its powerset, and the free guarded algebra is given by the partial maps functor $(\bot + (-))^{At}$ (where $At$ is the set of atoms of the Boolean algebra of tests). We call program branching that is determined by a set of equational laws *effectful*, in reference to Plotkin and Power's *algebraic effects* [PP02].

In op. cit., Plotkin and Power use equational theories in various categories to capture the monadic semantics of computational effects due to Moggi [Mog91]. Moggi uses so-called *strong* monads to give a semantics to program effects like exceptions, nondeterminism, probability, input and output, side-effects, and continuations. As Plotkin and Power show, many of these effects (excluding continuations) can be captured by a set of algebraic operations and equations acting on an object in some category [PP02]. In the context of this thesis, the objects we deal with are sets. It is interesting to note that an algebraic presentation of a monad in the category of sets induces a strength for that monad, although we will not make explicit use of monad strengths at any point in the thesis.

In this thesis, I use the term *effectful process algebra* to refer to any algebra of

[3]Although, Manes referred to algebras of if-then-else as *McCarthy algebras* [Man91].

abstract programs that exhibit a form of program branching that is determined by a set of equational laws. I introduce and develop the theory of effectful process algebra, in which GKAT, regular expressions modulo bisimilarity, and many other abstract programming languages can be studied uniformly and axiomatically. More specifically, GKAT and regular expressions modulo bisimilarity are exhibited as *star fragments*, fragments of larger calculi corresponding to each effectful branching type. I give a uniform complete axiomatization of these star fragments using a generalization of the *uniqueness axiom* of GKAT. Finally, encouraged by the positive results of Grabmayer [Gra22] and the partial completeness results for GKAT in Chapter 3, I conjecture that the uniqueness axiom can be derived from the other axioms for star fragments in general. This conjecture subsumes the (currently open) GKAT completeness problem, but widens the scope to other abstract programming languages, including several probabilistic languages.

## 1.3 Related Work

A detailed account of the literature related to the work in Chapter 2 can be found in the introduction to that chapter. For detailed accounts of the work related to Chapters 3 and 4, refer to Sections 3.6 and 4.6 respectively. For now, I briefly mention two important lines of research related to this thesis.

#### Kleene coalgebra

The effectful process algebra framework can be seen as a generalization of the calculi in Milner's paper [Mil84] to a much wider class of system types. In this sense, there are clear parallels between our work and the thesis of Silva [Sil10]. Silva's thesis introduces a family of calculi that includes certain effectful process algebras found in Chapter 4, but also includes many other system types (coalgebras for general polynomial functors on **Set**) that the effectful process algebra framework does not address. On the other hand, the effectful process algebra framework covers system types not included in Silva's framework: Effectful process algebras allow branching to be captured by a large class of equational theories, whereas Silva's is centred around one theory (semilattices).

Another, possibly more important difference between effectful process algebra and the calculi in Silva's thesis is that effectful process algebras do not always satisfy

an analogue of Kleene's theorem, whereas Silva's calculi do. The lack of expressiveness of most effectful process algebras makes for incredibly difficult completeness problems, nearly all of which are open.

The work [Mil10; BMS13] is also of note here. In op. cit., the authors generalize the coalgebraic calculi studied in Silva's thesis to categories other than **Set**. The calculi they obtain also satisfy a version of Kleene's theorem (replace *finite* with *finitely presentable*), which allows for their completeness proof to go through. Effectful process algebra can also be generalized to categories other than **Set** (see my preprint [Sch22a] for effectful process algebra with posets). The comparison between effectful process algebras in other categories and the calculi in [Mil10; BMS13] is analogous to the comparison with the calculi of Silva.

#### Iteration

In [Mil84], Milner remarks that a slight variation on his calculi "can probably... be fitted in to the framework of Elgot's [Elg75] iterative algebraic theories." Indeed, there is a notable connection between effectful process algebra and the iterative theories of Elgot [Elg75], Bloom and Ésik [BE76; BÉ88], Nelson [Nel83], and Adamék, Milius, and Velebil [AMV11]. Chapter 4, for example, uses what I call an *iterative branching theory*, a variation on the notion of Elgot monad from the last cited work. What sets effectful process algebra apart from existing work in this area is the flexibility of its syntax: Effectful process algebra handles recursion through single-variable fixed-points, which makes restrictions of axiomatizations to certain fragments straightforward. In iterative algebra, handling recursion when restricting to fragments requires characterizations of solvable systems of equations. These characterizations are often very difficult to obtain, which we can clearly see from the expressiveness problem posed by Milner and tackled in [BCG07].

## 1.4 Overview of the Thesis

The technical content of the thesis is broken up into three chapters, roughly covering (and improving on) four papers [Sch+21; SRS21; KSS23; Sch+22]. Aside from the use of coalgebraic tools, each of which I summarize in Chapter 2 (but can also be found in [Rut00; Gum99; Adá05; Jac16]), each chapter is self-contained. A breakdown of each chapter is given below.

**Chapter 2** This chapter introduces two methods for proving coalgebraic completeness theorems, called the *local* and *global methods*, that have their roots in the work of Jacobs [Jac06] and Silva [Sil10]. The guiding example of the proof method is a rephrasing of the completeness proof for the Horn axiomatization of *one-free* regular expressions modulo bisimilarity due to Grabmayer and Fokkink [GF20], which can be seen as an instance of the local method. The chapter begins with an introduction to Milner's completeness problem and gives an expository account of Grabmayer and Fokkink's proof in op. cit. Furthermore, a close examination of the global coalgebraic completeness proof method is used to expose a characterization of coalgebraic completeness proofs in general. This culminates in a completeness proof strategy (Lemma 2.5.3 and Theorem 2.3.2) that is used to prove the main completeness theorem in Chapter 4, Theorem 4.4.11.

This chapter is based on the paper *On Star Expressions and Coalgebraic Completeness Theorems* [SRS21], authored by myself, Jurriaan Rot, and Alexandra Silva, and presented by myself at MFPS in 2021. I was the main contributor to this work.

**Chapter 3** This chapter discusses the completeness problem of GKAT, the while fragment of KAT. Inspired by Grabmayer and Fokkink's partial solution to Milner's completeness problem, the completeness theorem for one-free regular expressions modulo bisimilarity [GF20], we introduce the *skip-free* fragment of GKAT and show that a subset of the axioms of GKAT give a complete axiomatization of bisimilarity in this fragment. The completeness proof is a reduction to the completeness of one-free regular expressions modulo bisimilarity. We furthermore show that the completeness of skip-free GKAT modulo guarded language equivalence can be reduced to the completeness of skip-free GKAT modulo bisimilarity.

Section 3.1 is loosely based on *Guarded Kleene Algebra with Tests: Coinduction, Coequations, and Completeness* [Sch+21], authored by myself, Tobias Kappé, Dexter Kozen, and Alexandra Silva, and presented by myself at ICALP 2021. I was the main contributor to this work.

The rest of the chapter is based on *A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests* [KSS23], authored by Tobias Kappé, myself, and Alexandra Silva, and presented by myself at ESOP 2023.

**Chapter 4** In this chapter, I observe that GKAT and regular expressions modulo bisimilarity are effectful process algebras, that they specify systems with branching type captured by equational laws. More specifically, I introduce an expressive calculus for each effectful branching type, along with both an operational and denotational semantics. I show that the two semantics coincide, and give a complete axiomatization of semantic equivalence. GKAT and regular expressions modulo bisimilarity are exhibited as certain fragments of these expressive calculi, which I call *star fragments*. For the cases where the free algebra construction satisfies a mild constraint, which includes GKAT automata and labelled transition systems, a complete axiomatization of the corresponding star fragment is obtained using a generalization of GKAT's uniqueness axiom.

This chapter is based on and improves on the paper *Processes Parametrized by an Algebraic Theory* [Sch+22], authored by myself, Wojciech Rozowski, Jurriaan Rot, and Alexandra Silva, and presented by myself at ICALP 2022. I was the main contributor to this paper. The chapter generalizes the work in [Sch+22] by including the notion of *variable reduction operator* taken from my preprint *A (Co)Algebraic Framework for Ordered Processes* [Sch22a].

**Chapter 5** In this chapter, I speculate on the future of effectful process algebra. The conjectures found in the technical chapters above are summarized, and an overview of currently ongoing work is presented.

## Chapter 2

# Star Expressions and Coalgebraic Completeness Theorems

In 1984, Robin Milner gave a non-standard interpretation of regular expressions [Mil84], viewing them as behaviours of nondeterministic processes rather than languages recognized by finite automata. This affects the semantics in two key ways: First, there are finite nondeterministic processes that do not behave like any regular expression[1]. This is in stark contrast with the language semantics of regular expressions, where Kleene's theorem tells us that every finite automaton has a language equivalent regular expression [Kle56]. Second, there are axioms in Salomaa's complete axiomatization of the algebra of regular languages [Sal66] that are unsound in the process interpretation. Milner acknowledges these changes in [Mil84] and outlines a modified version of Salomaa's axiomatization that is sound in the new interpretation. He does not offer a proof of completeness, however, and leaves it as an open problem.

Thirty-eight years later, Milner's completeness problem was solved by Clemens Grabmayer [Gra22], who extended a partial solution to the problem achieved earlier in joint work with Wan Fokkink [GF20]. The present chapter contains a description of Milner's completeness problem, but is more specifically concerned with analyzing the completeness proof of Grabmayer and Fokkink [GF20]. Grabmayer and Fokkink's completeness proof does not explicitly make use of coalgebra, but as we will see, its overall structure can be rephrased in the language of coalgebra. It is our motivating example of a *coalgebraic completeness theorem*, a concept I introduce in this chapter.

[1] A characterization of those that do was found in [BCG07].

Like many related completeness proofs that go through automata or other types of transition systems, Grabmayer and Fokkink's completeness proof consists of four key parts. The first is the production of models from expressions through the operational semantics. The second is a sort of inverse to the first, a notion of *solution to a model* in the class of expressions. The third is the identification of a distinguished class of models that captures the semantics of the expressions, every member of which admits a *unique* solution modulo the axioms. This tells us that every model in the distinguished class corresponds to an expression modulo the axioms. The fourth is the ability to reduce equivalent pairs of models to a common representative without leaving the distinguished class. This allows us to identify expressions up to provable equivalence if they are assigned equivalent models.

The third and fourth parts are subtler than the first two. In a classical proof such as Salomaa's [Sal66], but also in more recent coalgebraic formulations (for example, [Sil10; Jac06]), the distinguished class typically consists of all finite (or *locally finite*) automata. In this setting, solvability of finite systems can be done inductively on the number of free variables, and comparing automata simply consists of finding a bisimulation between them. Bisimulations between finite automata can always be equipped with the structure of a finite automaton, so the fourth step is rarely worth mentioning in this situation. Historically, the highest hurdle to clear has been the issue of proving that solutions are unique (in [Sal66], for example). In the setting of Grabmayer and Fokkink's paper [GF20], on the other hand, the class of models considered consists of the so-called *LLEE-charts*. The need for such a non-trivial class comes from the above-mentioned issue that there are finite processes that are not bisimilar to any regular expression. For LLEE-charts, uniqueness of solutions is not a triviality, but also does not warrant a proof in the main body of [GF20]. In comparison, a great amount of ingenuity goes into establishing the fourth of the proof steps mentioned above: The ability to reduce equivalent LLEE-charts to a common LLEE-chart.

Grabmayer and Fokkink's completeness proof is highly innovative and technical, and makes use of new tools carefully crafted for proving the compositionality result mentioned above. The abstract view I present here is no replacement for the detailed combinatorial arguments found in [GF20]. Instead, my intent is to unpack its

contents by situating them in the context of coalgebra [Rut00; Jac16]. Coalgebra is a well-established general framework for state-based systems that subsumes constructs like bisimilarity and behaviour and provides us with a convenient language for generalizing the important conceptual details in the completeness proof of [GF20].

After a brief coalgebra-focused introduction to Milner's star expressions, I turn to the paper of Grabmayer and Fokkink [GF20] to give a coalgebraic spin on some of their results. I strengthen some and simplify the proofs of others:

- I show that one-free regular expressions form a coalgebra, and that solutions to transition systems are in one-to-one correspondence with coalgebra homomorphisms into the expressions modulo the axioms (Lemma 2.2.17).
- I elucidate the four steps in Grabmayer and Fokkink's completeness proof mentioned above and prove they are sufficient in a general coalgebraic setting.
- I generalize the *connect-through-to* operation from [GF20] to a purely coalgebraic construction (Definition 2.4.14). I coin the term *rerouting* for this construction, and show that a prevalence of reroutings can be used to establish the fourth part of completeness proofs (Lemma 2.4.15).
- Finally, I give a general account of a related approach to completeness proofs that originates in Jacobs's bialgebraic reformulation [Jac06] of Kozen's completeness proof for Kleene algebra [Koz91] and the more general completeness theorems Jacobs's technique inspired [Sil10; Mil10; BMS13]. I furthermore show that the proof technique of Grabmayer and Fokkink can be restructured to fit this mould.

Overall, I use the structure of the completeness proof in [GF20] as a case study in completeness proof methods from coalgebra that do not rely on a surjective correspondence between expressions and finite automata up to bisimulation. This culminates in two abstract proof methods for completeness, what I call the *local* and *global* approaches, and a description of those situations in which the latter method can be used in place of the former.

The chapter is organized as follows: In Section 2.1, I introduce Milner's reinterpretation of regular expressions and state the completeness result recently announced by Grabmayer [Gra22]. In Section 2.2, I introduce the one-free fragment of regular expressions in parallel with its coalgebraic aspects. In Section 2.3, I discuss

the four parts of Grabmayer and Fokkink's completeness proof and show that they are sufficient in a general coalgebraic setting. In Section 2.4, I give an alternative description of LLEE-charts and show how Grabmayer and Fokkink's technique for reducing LLEE-charts can be strengthened. It is in this section that I generalize their *connect-through-to* operation. Lastly, in Section 2.5, I give a general account of a related approach to completeness proofs, found in [Jac06; Sil10; Mil10; BMS13], and show how the method used by Grabmayer and Fokkink can be restructured to fit this mould.

> The contents of this chapter, excluding Section 2.1, are based on the paper *On Star Expressions and Coalgebraic Completeness Theorems*, written by myself, Jurriaan Rot, and Alexandra Silva [SRS21].

## 2.1 Star Expressions and Grabmayer's Theorem

In a technical report from 1951, Stephen Cole Kleene gave an early example of a finite syntax for specifying behaviours of finite state machines [Kle56]. The terms of his syntax are the well-renowned *regular expressions*, formed by the grammar

$$RExp \ni r, r_1, r_2 := 0 \mid 1 \mid a \in Act \mid r_1 + r_2 \mid r_1 r_2 \mid r^*$$

where $Act$ is a given set of *action symbols*. Regular expressions were intended to represent what Kleene called *regular events*, what are now called *regular languages*.

*Remark* 2.1.1. Actually, in the original paper [Kle56], the syntax Kleene used to describe regular expressions had a binary star operation ${r_1}^* r_2$. The binary star appears to have fallen out of fashion after Kleene's publication—see, for example, [Sal66; Con71]. This is due in part to the fact that the binary star gives an equivalent set of expressions to the unary star, as $r^*1$ and $r^*$ are semantically equivalent. However, the binary star will reappear in Section 2.2 when we consider the set of terms generated without the use of 1 (reintroduced by [GF20]).

**Definition 2.1.2.** Fix a set $Act$ of action symbols. A *language* is a subset of the free monoid $Act^*$ generated by $Act$. The language $\mathcal{L}(r)$ *denoted* by a regular expression $r \in RExp$ is defined inductively as

$$\mathcal{L}(0) = \emptyset \qquad \mathcal{L}(1) = \{\varepsilon\} \qquad \mathcal{L}(r_1 + r_2) = \mathcal{L}(r_1) \cup \mathcal{L}(r_2)$$

$$\mathcal{L}(r_1 r_2) = \mathcal{L}(r_1)\mathcal{L}(r_2) := \{w_1 w_2 \mid w_i \in \mathcal{L}(r_i)\}$$
$$\mathcal{L}(r^*) = \mathcal{L}(r)^* := \{\varepsilon\} \cup \mathcal{L}(r) \cup \mathcal{L}(r)\mathcal{L}(r) \cup \cdots$$

A language is *regular* if it of the form $\mathcal{L}(r)$ for some regular expression $r$. Two regular expressions $r_1, r_2$ are called *language equivalent* if $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.

For example, $r_1 + r_2$ and $r_2 + r_1$ are always language equivalent, no matter our choice of $r_1, r_2$, and so are $r_1(r_2 + r_3)$ and $r_1 r_2 + r_1 r_3$. Kleene noticed that equations like "$r_1 + r_2 = r_2 + r_1$" and "$r_1(r_2 + r_3) = r_1 r_2 + r_1 r_3$" hold for the language interpretation, but left axiomatization as an open problem.

**Definition 2.1.3.** Given a set of inference rules $\mathsf{TH}$ for deriving equations between terms in an algebraic signature $S$ and a pair of expressions $r_1, r_2$, write $\mathsf{TH} \vdash r_1 = r_2$ if $r_1 = r_2$ is the conclusion of a proof using only the rules in $\mathsf{TH}$ and the axioms of equational logic,

$$\begin{array}{llll} \text{(Ref)} & \text{(Sym)} & \text{(Tra)} & \text{(Con)} \\ r = r & \dfrac{r_2 = r_1}{r_1 = r_2} & \dfrac{r_1 = r_3 \quad r_3 = r_2}{r_1 = r_2} & \dfrac{(\forall i \leq n)\ r_i = r_i'}{\sigma(r_1, \ldots, r_n) = \sigma(r_1', \ldots, r_n')} \end{array}$$

where $\sigma$ is an $n$-ary operation from $S$. We often refer to $\mathsf{TH}$ as an *axiomatization*, *inference system*, or *theory*.

Regular expressions are terms in the algebraic signature $S = Act \cup \{0, 1, +, \cdot, (-)^*\}$, where each $a \in Act$ and 0 and 1 are *constants* (0-ary operations), $+$ and $\cdot$ (representing sequential composition) are binary operations, and $(-)^*$ is a unary operation.

**Definition 2.1.4.** A function of the form $[\![-]\!] : RExp \to Z$ is called a *semantics* of *RExp*, $Z$ the *semantic domain*, and we say $r_1$ and $r_2$ are $[\![-]\!]$*-equivalent* if $[\![r_1]\!] = [\![r_2]\!]$. An axiomatization $\mathsf{TH}$ is *sound* for $[\![-]\!]$ if $\mathsf{TH} \vdash r_1 = r_2$ implies $[\![r_1]\!] = [\![r_2]\!]$, and *complete* for $[\![-]\!]$ if $[\![r_1]\!] = [\![r_2]\!]$ implies $\mathsf{TH} \vdash r_1 = r_2$. Given an equivalence relation $\equiv$ on *RExp*, we say $\mathsf{TH}$ is *sound/complete* with respect to $\equiv$ if $\mathsf{TH}$ is sound/complete with respect to the quotient semantics $[-]_{\equiv} : RExp \to RExp/\equiv$.

Kleene posed the problem of finding a sound and complete axiomatization of language equivalence in [Kle56]. The first complete axiomatization was found by Salomaa [Sal66]. We reproduce Salomaa's axiomatization in Figure 2.1, where

| | | | |
|---|---|---|---|
| (B0) | $r+r=r$ | (FP1) | $r_1{}^*r_2 = r_1(r_1{}^*r_2)+r_2$ |
| (B1) | $r+0=r$ | (FP2) | $(r_1+1)^* = r_1{}^*$ |
| (B2) | $r_1+r_2=r_2+r_1$ | (RSP*) | $\dfrac{s=r_1s+r_2 \quad E(r_1)=0}{s=r_1{}^*r_2}$ |
| (B3) | $r_1+(r_2+r_3)=(r_1+r_2)+r_3$ | | |
| (B4) | $r1=r$ | (A1) | $r0=0$ |
| (B5) | $1r=r$ | (A2) | $r_1(r_2+r_3)=r_1r_2+r_1r_3$ |
| (B6) | $0r=0$ | | |
| (B7) | $r_1(r_2r_3)=(r_1r_2)r_3$ | | |
| (B8) | $(r_1+r_2)r_3=r_1r_3+r_2r_3$ | | |

**Figure 2.1:** Salomaa's complete axiom system for language equivalence of regular expressions. Here, $r,r_1,r_2,s \in \mathit{RExp}$, $a \in \mathit{Act}$, and $\star \in \{+, {}^*\}$. The system Mil consists of (B0)-(B8), (FP1), (FP2), and (RSP*).

special attention is drawn to the axioms (A1) and (A2). In the rule (RSP*) from Figure 2.1, we make use of a predicate $E$, defined

$$E(0)=E(a)=0 \qquad E(1)=E(r^*)=1$$
$$E(r_1+r_2)=E(r_1)\vee E(r_2) \qquad E(r_1r_2)=E(r_1)\wedge E(r_2)$$

for any $a \in \mathit{Act}$ and $r,r_1,r_2 \in \mathit{RExp}$. We say that $r$ is *productive* if $E(r)=0$, and if $E(r)=1$ we say that $r$ has the *empty word property*.

In the late 60s, regular expressions became the basis for string searching algorithms [Tho68]. When interpreted this way, regular expressions should be identified up to language equivalence, because that is the equivalence that best captures their computational value in the context of string searching. However, in many other applications, particularly those that involve interaction during runtime, language equivalence identifies too many regular expressions. Consider, for example, the pair of vending machines in Figure 2.2.

In the first vending machine, upon receiving a two-dollar coin, the machine presents its user with the options of juice and coffee. The second vending machine is far more disappointing: Upon receiving a two-dollar coin, it either enters a state where the user can only choose orange juice or a state where the user can only choose coffee. Everybody likes to have decisions made for them once in a while, but I would not purchase the second machine for my office!

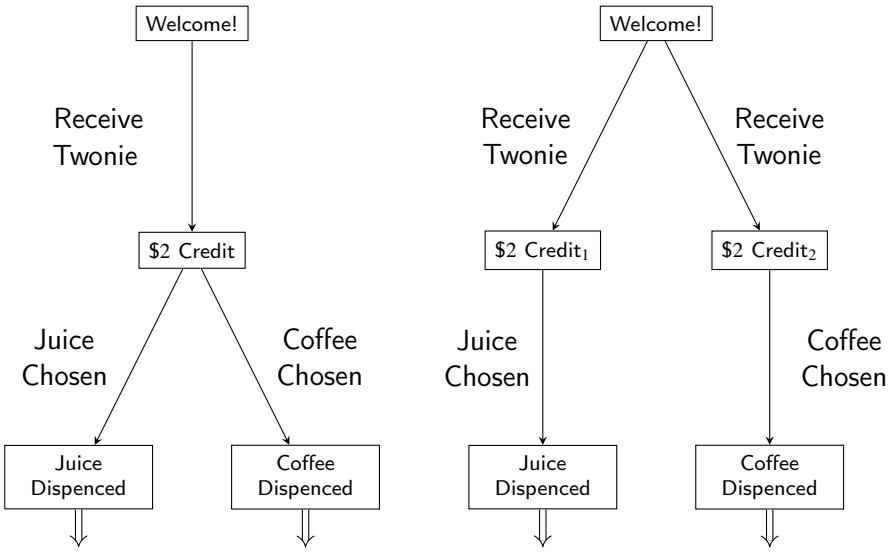



**Figure 2.2:** Two vending machines.

In Figure 2.2, a labelled arrow $x \xrightarrow{a} y$ represents an action $a$ that induces a transition between states $x$ and $y$ of the vending machine. Interactions with either machine start with a welcome message, and states where interactions with the machine are allowed to end appear as the source of a double arrow. Both machines are allowed to end precisely after one of the runs of actions in the language below

$$\{(\text{Receive Twonie})(\text{Juice Chosen}), (\text{Receive Twonie})(\text{Coffee Chosen})\} \tag{2.1}$$

That is, they recognize the same language. While they recognize the same language, they are certainly not equivalent in this context. The notion that captures the appropriate equivalence in this context is *bisimilarity* [Mil80], and can be formalized with *labelled transition systems*.

**Definition 2.1.5.** A *labelled transition system* (or *LTS*) is a pair $(X, \delta)$ consisting of a set $X$ and a function $\delta \colon X \longrightarrow \mathcal{P}(\checkmark + Act \times X)$. Here, $\checkmark$ is shorthand for the set $\{\checkmark\}$. We write $x \xrightarrow{a} y$ when $(a, y) \in \delta(x)$ and $x \Rightarrow$ when $\checkmark \in \delta(x)$.

In a LTS $(X, \delta)$, the language $\mathcal{L}_\delta(x)$ *recognized by a state* $x \in X$ is formally defined

$$\mathcal{L}_\delta(x) = \left\{ a_1 \cdots a_n \in Act^* \mid (\exists n \in \mathbb{N})(\exists x_1, \ldots, x_n)\ x \xrightarrow{a_1} x_1 \to \cdots \to x_{n-1} \xrightarrow{a_n} x_n \Rightarrow \right\}$$

When $\mathcal{L}_\delta(x) = \mathcal{L}_\delta(y)$, we say $x$ and $y$ are *language equivalent*. For example, (2.1) is

$$1 \Rightarrow \qquad \frac{a \in Act}{ar \xrightarrow{a} r} \qquad \frac{r_1 \Rightarrow}{r_1 + r_2 \Rightarrow} \qquad \frac{r_2 \Rightarrow}{r_1 + r_2 \Rightarrow} \qquad \frac{r_1 \xrightarrow{a} s}{r_1 + r_2 \xrightarrow{a} s} \qquad \frac{r_2 \xrightarrow{a} s}{r_1 + r_2 \xrightarrow{a} s}$$

$$\frac{r_1 \Rightarrow \quad r_2 \Rightarrow}{r_1 r_2 \Rightarrow} \qquad \frac{r_1 \Rightarrow \quad r_2 \xrightarrow{a} s}{r_1 r_2 \xrightarrow{a} s} \qquad \frac{r_1 \xrightarrow{a} s}{r_1 r_2 \xrightarrow{a} s r_2} \qquad r^* \Rightarrow \qquad \frac{r \xrightarrow{a} s}{r^* \xrightarrow{a} s r^*}$$

**Figure 2.3:** The LTS structure $\ell \colon RExp \to \mathcal{P}(\checkmark + Act \times RExp)$. Here, $r, r_1, r_2, s \in RExp$.

$\mathcal{L}_\delta(\text{Welcome!})$ for both Welcome! states, so the two states are language equivalent. Bisimilarity, a more discerning equivalence, is formally defined below.

**Definition 2.1.6.** A *bisimulation* $R$ between LTSs $(X, \delta_X)$ and $(Y, \delta_Y)$ is a relation $R \subseteq X \times Y$ such that for any $(x, y) \in R$,

(i) $x \Rightarrow$ if and only if $y \Rightarrow$,

(ii) if $x \xrightarrow{a} x'$, then there is a $y' \in Y$ such that $y \xrightarrow{a} y'$ and $(x', y') \in R$,

(iii) if $y \xrightarrow{a} y'$, then there is an $x' \in X$ such that $x \xrightarrow{a} x'$ and $(x', y') \in R$.

We call $x$ and $y$ *bisimilar*, and write $x \leftrightarroweq y$, if $(x, y)$ appears in some bisimulation.

Bisimilarity tracks not only what runs of the machine end in success, like language equivalence does, but also the branching structure of the machine's behaviour. Bisimilarity implies language equivalence, but language equivalence does not imply bisimilarity. For example, in Figure 2.2, the state "\$2 Credit$_1$" cannot take the "Juice Chosen" action, and this prevents the two vending machines from being bisimilar.

Now let us turn to the issue of deciding which pairs of regular expressions should be bisimilar. Following Antimirov [Ant96], we equip $RExp$ with the inductively defined transition function $\ell \colon RExp \to \mathcal{P}(\checkmark + Act \times RExp)$ found in Figure 2.3. Given $e \in RExp$, we write $\langle e \rangle$ for the LTS generated by the expressions reachable from $e$ and call it the *small-step semantics* of $e$.

**Lemma 2.1.7.** *Let $r_1, r_2$ be regular expressions.*

1. *[Ant96] $\mathcal{L}(r_1) = \mathcal{L}_\ell(r_1)$. That is, the language semantics of regular expressions coincides with the language accepted by regular expressions as states in $(RExp, \ell)$.*
2. *[Ant96] $\langle r \rangle$ has finitely many states.*
3. *[Ant96] If $r_1 \leftrightarroweq r_2$, then $\mathcal{L}(r_1) = \mathcal{L}(r_2)$.*

4. *[Mil84] Restricted to* $(RExp, \ell)$, $\leftrightarrows$ *is a* congruence*, meaning if* $r_1 \leftrightarrows s_1$ *and* $r_2 \leftrightarrows s_2$*, then* $r_1 + r_2 \leftrightarrows s_1 + s_2$ *and* $r_1{}^* s_1 \leftrightarrows r_2{}^* s_2$.

For example, the vending machines in Figure 2.2 are the small-step semantics of

$$(\text{Receive Toonie})((\text{Juice Chosen}) + (\text{Coffee Chosen}))$$
$$(\text{Receive Toonie})(\text{Juice Chosen}) + (\text{Receive Toonie})(\text{Coffee Chosen})$$

respectively. As was noted before, these are *not* bisimilar machines, despite their equality being an instance of (A2) in Figure 2.1. It is also easy to find a pair of expressions that are an instance of (A1) and yet not bisimilar: take $a0$ and $0$, for example. This is all to say that axioms (A1) and (A2) are not sound with respect to bisimilarity, although the rest of the axioms in Figure 2.1 are. Following [Gra21], write Mil for the system consisting of (B0)-(B8), (FP), and (RSP*). The following result says that Mil is a sound axiomatization of bisimilarity.

**Theorem 2.1.8** (Soundness [Mil84])**.** *Let* $r_1, r_2 \in RExp$*. If* $\mathsf{Mil} \vdash r_1 = r_2$*, then* $r_1 \leftrightarrows r_2$.

In [Mil84], Milner furthermore asks whether Mil is *complete* with respect to bisimilarity, i.e., whether $r_1 \leftrightarrows r_2$ implies $\mathsf{Mil} \vdash r_1 = r_2$. This question I will refer to as *Milner's completeness problem*. After 38 years of research into Milner's completeness problem, a solution was recently announced by Grabmayer, whose proof builds on earlier work by himself and Fokkink [GF20].

**Theorem 2.1.9** (Completeness [Gra22])**.** *Let* $r_1, r_2 \in RExp$*. If* $r_1 \leftrightarrows r_2$*, then we can prove* $\mathsf{Mil} \vdash r_1 = r_2$.

Salomaa's completeness proof [Sal66] was well-known to Milner and many others from the process algebra community when [Mil84] was written. Furthermore, Milner essentially follows Salomaa's proof to establish a different (but related) completeness theorem in the same paper [Mil84]. Why did it take so long for a proof of Theorem 2.1.9 to appear?

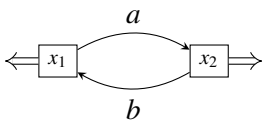
Salomaa's completeness proof relies on one's ability to solve recursive systems of equations in the algebra of regular languages. Consider, for example,

$$x_1 = ax_2 + 1 \qquad x_2 = bx_1 + 1 \tag{2.2}$$

Solving this system algebraically is a matter of plugging the second equation into the first and using (A2) and (RSP*) to turn $x_1 = a(bx_1 + 1) + 1$ into $x_1 = abx_1 + (a + 1)$ and then into $x_1 = (ab)^*(a + 1)$. The last equation tells us that assigning $x_1$ the language $\{ab\}^* \cdot \{\varepsilon, a\}$ and $x_2$ the language $\{b\} \cdot (\{ab\}^* \cdot \{\varepsilon, a\}) \cup \{\varepsilon\}$ solves the system (2.2) in the algebra of regular languages. This overall method of solving equations is behind the following theorem.

**Theorem 2.1.10.** *Let $L \subseteq Act^*$ be a language. Then $L$ is a regular language if and only if there is a state $x \in X$ in a finite LTS $(X, \delta)$ such that $L = \mathcal{L}_\delta(x)$.*

Theorems like Theorem 2.1.10 are known as *Kleene theorems*, and as we will see in the following pages, they are central to completeness results. Unfortunately, there are examples of LTSs with states that are not bisimilar to any regular expression in $(RExp, \ell)$. Consider again (2.2), which corresponds to the LTS




(2.3)

As it turns out, a simple inductive argument shows that neither $x_1$ nor $x_2$ in (2.3) are bisimilar to any regular expression in $(RExp, \ell)$. We include a proof below, although this was known to Milner [Mil84] and it follows from a general characterization of LTSs bisimilar to regular expressions due to Baeten et al. [BCG07].

**Proposition 2.1.11.** *No state of $(RExp, \ell)$ is bisimilar to either $x_1$ or $x_2$ in (2.3).*

*Proof.* We show by induction on $r \in StExp$ that $\neg(x_i \leftrightarrows r')$ for any $r' \in \langle r \rangle$ and $i \in \{1, 2\}$. The base cases are clear, since if $x_i \leftrightarrows r'$ for $r \in \{0, 1\} \cup Act$, then $\langle x_i \rangle$ would be acyclic.

For the first inductive step, assume $\neg(x_i \leftrightarrows s)$ for any $s \in \langle r_j \rangle$ for $i, j \in \{1, 2\}$. To see that $x_1$ is not bisimilar to any $s \in \langle r_1 + r_2 \rangle$, we need only consider the $s = r_1 + r_2$ case, since every other $s \in \langle r_1 + r_2 \rangle$ is in either $\langle r_1 \rangle$ or $\langle r_2 \rangle$. If $x_1 \leftrightarrows r_1 + r_2$, then $r_j \xrightarrow{a} r'$ for some $j \in \{1, 2\}$ and some $r'$ with $r' \leftrightarrows x_2$. This contradicts the induction hypothesis. Similarly for the $x_2$ case.

Now suppose $x_1 \leftrightarrows s$ for some $s \in \langle r_1 r_2 \rangle$. Either $s = r_1' r_2$ for some $r_1' \in \langle r_1 \rangle$ or $s \in \langle r_2 \rangle$. We have excluded the latter situation by assumption, so consider the former. Without loss of generality, we can assume $x_1 \leftrightarrows r_1 r_2$, because $\langle r_1' \rangle \subseteq \langle r_1 \rangle$. We are going

to generate a contradiction by observing that $x_1 \leftrightarrows r_1 r_2$ implies that either $x_2 \leftrightarrows r'$ for some $r' \in \langle r_2 \rangle$ or $x_1 \leftrightarrows r_1$.

Since $x_1 \Rightarrow$ and $x_1 \leftrightarrows r_1 r_2$, $r_1 \Rightarrow$ also. So, if $r_2 \xrightarrow{a} s$ for some $s \in \langle r_2 \rangle$, then $s \leftrightarrows x_2$, because $r_1 r_2 \xrightarrow{a} s$ and $x_1 \leftrightarrows r_1 r_2$. This contradicts the induction hypothesis, so it must be the case that $\neg(r_2 \xrightarrow{a})$. Symmetrically, if $r_1 \xrightarrow{a} r'$, then $x_2 \leftrightarrows r' r_2$ and $x_1 \Rightarrow$, so $r' \Rightarrow$ as well. Again, if $r_2 \xrightarrow{b} s$, then $r' r_2 \xrightarrow{b} s$ and $x_1 \leftrightarrows s$ for some $s \in \langle r_2 \rangle$, so it must also be the case that $\neg(r_2 \xrightarrow{b})$.

We are now going to use $\neg(r_2 \xrightarrow{a})$ and $\neg(r_2 \xrightarrow{b})$ to show that $x_1 \leftrightarrows r_1$. In particular, we are going to prove that $R = \{x_1\} \times U \cup \{x_2\} \times V$ is a bisimulation, where

$$U = \{r'' \mid (\exists n)\ r_1 (\xrightarrow{a} \circ \xrightarrow{b})^n r''\} \qquad V = \{r' \mid (\exists n)\ r_1 \xrightarrow{a} \circ (\xrightarrow{b} \circ \xrightarrow{a})^n r'\}$$

Note that $(x_1, r_1) \in R$. Also note that an easy induction on $n$ reveals that $x_1 \leftrightarrows r'' r_2$ and $x_2 \leftrightarrows r' r_2$ for any $r'' \in U$ and $r' \in V$. To see that $R$ is a bisimulation, observe that $x_1 \xrightarrow{a} x_2$ and $x_1 \leftrightarrows r'' r_2$ and $\neg(r_2 \xrightarrow{a})$ imply that $r'' \xrightarrow{a} r'$ for some $r' \in V$. Conversely, if $U \ni r'' \xrightarrow{c} r'$, then $c = a$ because $x_1 \leftrightarrows r'' r_2$, and therefore $r' \in V$. Since $(x_2, r') \in R$, we are done. In the symmetric case, $(x_2, r') \in R$ and $x_2 \xrightarrow{b} x_1$ imply there is a transition $r' \xrightarrow{b} r'' \in U$, since $x_2 \leftrightarrows r' r_2$. And similarly to the case for $x_1$, since $\neg(r_2 \xrightarrow{b})$, if $r' \xrightarrow{c} r''$, then $c = b$ and $r'' \in U$. It follows that $R$ is a bisimulation. Since $x_1 R r_1$, $x_1 \leftrightarrows r_1$. This contradicts the induction hypothesis.

For the last case, we begin by assuming $\neg(x_i \leftrightarrows s)$ for every $s \in \langle r \rangle$ and $i \in \{1,2\}$ to show that $\neg(x_1 \leftrightarrows r^*)$. Indeed, if $x_1 \leftrightarrows r^*$, then $r^* \xrightarrow{a} s r^*$ for some $s$ and $x_2 \leftrightarrows s r^*$. However, since $x_2 \Rightarrow$, $s \Rightarrow$ as well. This means that $s r^* \xrightarrow{a}$, which contradicts $\neg(x_2 \xrightarrow{a})$. Hence, $\neg(x_1 \leftrightarrows r^*)$. Symmetrically, $\neg(x_2 \leftrightarrows r^*)$.

Finally, assume $\neg(x_i \leftrightarrows s)$ for every $s \in \langle r \rangle$ and $i \in \{1,2\}$. We suppose for a contradiction that there is an $s \in \langle r \rangle$ such that $x_1 \leftrightarrows s r^*$. Since $x_1 \Rightarrow$ and $\neg(x_1 \xrightarrow{b})$, $\neg(r^* \xrightarrow{b})$ by the transition rules for sequential composition. Similarly, since for any transition $s r^* \xrightarrow{a} s' r^*$ we find $x_2 \leftrightarrows s' r^*$, $\neg(r^* \xrightarrow{a})$.

We are now going to use $\neg(r^* \xrightarrow{a})$ and $\neg(r^* \xrightarrow{b})$ to establish that the relation $R = \{x_1\} \times U \cup \{x_2\} \times V$ is a bisimulation, where

$$U = \{s'' \mid (\exists n)\ s (\xrightarrow{a} \circ \xrightarrow{b})^n s''\} \qquad V = \{s' \mid (\exists n)\ s \xrightarrow{a} \circ (\xrightarrow{b} \circ \xrightarrow{a})^n s'\}$$

Note that $(x_1, s) \in R$ and that an easy induction on $n$ establishes that $x_1 \leftrightarrows s''$ and $x_2 \leftrightarrows s'$ for any $s'' \in U$ and $s' \in V$. Given $(x_1, s'') \in R$, since $x_1 \leftrightarrows s''r^*$ and $\neg(r^* \xrightarrow{a})$, $s'' \xrightarrow{a} s'$ for some $s' \in V$. Conversely, if $s'' \xrightarrow{c} s'$, then $c = a$ and therefore $s' \in V$ as desired. Similarly for any pair $(x_2, s') \in R$ (the details are similar to the sequential composition case). This establishes that $R$ is a bisimulation, so $x_1 \leftrightarrows s$. Since $s \in \langle r \rangle$, this contradicts the induction hypothesis. □

Examples like (2.3) expose the central difficulty of the proof of Theorem 2.1.9, which requires an intricate characterization of the systems of equations that *can* be solved. Theorem 2.1.9 is an incredible achievement that is likely to have applications to open completeness problems like the ones found in Chapters 3 and 4. The promise of these potential applications is strengthened in the next section, which sets the stage for the rest of the thesis by analyzing the joint work of Grabmayer and Fokkink [GF20] that proves a version of Theorem 2.1.9 for a fragment of *RExp*. While Grabmayer and Fokkink's paper presents only a partial solution to Milner's completeness problem, their completeness proof is our first clear example of a coalgebraic completeness proof. Their proof strategy is an instance of a general coalgebraic strategy for proving completeness theorems that will be introduced in the next section.

## 2.2 One-free Star Expressions

Grabmayer's proof of Theorem 2.1.9 relies on earlier work with Fokkink [GF20] that presents a partial solution to Milner's completeness problem, namely a proof that if $r_1 \leftrightarrows r_2$ and $r_1, r_2$ are *one-free*, then $\mathsf{Mil} \vdash r_1 = r_2$. For the rest of this chapter, we define what it means for a regular expression to be *one-free* and connect the study of one-free regular expressions to the theory of universal coalgebra.

For a fixed finite set $Act$ of *atomic actions*, the set of *one-free star expressions*, or *star expressions* for short, is generated by the BNF grammar

$$StExp \ni r_1, r_2 ::= 0 \mid a \in Act \mid r_1 + r_2 \mid r_1 r_2 \mid r_1 * r_2$$

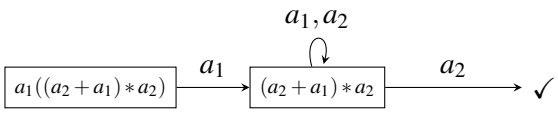
The expression $r_1 * r_2$ denotes the regular expression $r_1{}^* r_2$ from the previous section (see also [Mil84]), but we write $*$ infix to emphasize that it is a binary operation in this formalism (like in Kleene's seminal paper [Kle56]). Operationally, each star expression specifies a certain kind of labelled transition system with termination.

**Definition 2.2.1.** A *one-free transition system* is a pair $(X,\delta)$ consisting of a set $X$ of *states* and a *transition function* $\delta\colon X \to \mathcal{P}(Act \times (\checkmark + X))$. Again, we write $x \xrightarrow{a} y$ if $(a,y) \in \delta(x)$, and we write $x \xrightarrow{a} \checkmark$ if $(a,\checkmark) \in \delta(x)$. A *prechart* is a one-free transition system $(X,\delta)$ that is *finitely branching*, meaning that $\delta(x)$ is finite for all $x \in X$.

*Remark* 2.2.2. The difference between labelled transition systems and one-free transition systems is the existence of a designated termination "state": In labelled transition systems, any state can terminate a computation by including a $x \Rightarrow$ transition, but in one-free transition systems only the $\checkmark$ "state" can end a computation. I put quotations around the word "state" because $\checkmark \notin X$, so technically $\checkmark$ is not a state. However, every one-free transition system can be made into a labelled transition system by creating a new state $\boxed{\checkmark}\Rightarrow$.

The operational interpretation of a star expression is obtained by giving the set of star expressions a structure equivalent to that of a prechart $(StExp,\ell)$. The transitions of $(StExp,\ell)$ are built inductively from the interpretations of expressions as processes: The constant 0 is deadlock, $a \in Act$ is the process that performs the action $a$ and then terminates, $r+s$ and $rs$ are alternative and sequential composition respectively, and $r*s$ iterates $r$ before executing $s$. Formally, the transitions of $(StExp,\ell)$ are those derivable from the rules in Figure 2.4.

*Example* 2.2.3. For example, the prechart $\langle a_1((a_2+a_1)*a_2)\rangle$ is




**Definition 2.2.4.** Let $(X,\delta)$ be a prechart and $U \subseteq X$. We define

$$U' = \{x \in X \mid (\exists y \in U)(\exists a_0,\dots,a_n \in Act)(\exists y_1,\dots,y_n)\ y \xrightarrow{a_0} y_1 \to \cdots \to y_n \xrightarrow{a_n} x\}$$

and set $\langle U\rangle_\delta = (U', \delta_{U'})$, where $\delta_{U'}$ is induced by restricting the transition relations $\xrightarrow{a}$ to $U'$. A *chart* is a prechart of the form[2] $\langle x\rangle_\delta$ for some $x \in X$. Given a star expression $r \in StExp$, the *chart interpretation* or *small-step semantics* of $r$ is $\langle r\rangle_\ell$. We generally omit the transition function and write $\langle x\rangle$ or $\langle r\rangle$ when it is implicit.

[2]It would be more precise to write $\langle\{x\}\rangle_\delta$, but we generally identify an element $x$ with its one-element set $\{x\}$ throughout the thesis.

$$\frac{a \in Act}{a \xrightarrow{a} \checkmark} \qquad \frac{r_1 \xrightarrow{a} \checkmark}{r_1 + r_2 \xrightarrow{a} \checkmark} \qquad \frac{r_2 \xrightarrow{a} s}{r_1 + r_2 \xrightarrow{a} s} \qquad \frac{r_1 \xrightarrow{a} \checkmark}{r_1 r_2 \xrightarrow{a} r_2} \qquad \frac{r_1 \xrightarrow{a} s}{r_1 r_2 \xrightarrow{a} s r_2}$$

$$\frac{r_2 \xrightarrow{a} \checkmark}{r_1 * r_2 \xrightarrow{a} \checkmark} \qquad \frac{r_2 \xrightarrow{a} s}{r_1 * r_2 \xrightarrow{a} s} \qquad \frac{r_1 \xrightarrow{a} s}{r_1 * r_2 \xrightarrow{a} s(r_1 * r_2)} \qquad \frac{r_1 \xrightarrow{a} \checkmark}{r_1 * r_2 \xrightarrow{a} r_1 * r_2}$$

**Figure 2.4:** The prechart $(StExp, \ell)$. Here, $s, r_i \in StExp$ and $a \in Act$.

The small-step semantics of a star expression tracks many details about runtime that are not usually of any concern to the user. For example, the observable behaviours of $a(b+c)+a(c+b)$ and $a(b+c)$ are the same: They both consist of an emitted $a$ followed by the option of emitting $b$ or $c$. However, $\langle a(b+c)+a(c+b)\rangle$ and $\langle a(b+c)\rangle$ are not equal (or even isomorphic)! Semantically, we would like to identify $a(b+c)+a(c+b)$ and $a(b+c)$ while keeping track of the kinds of interactions we saw in Figure 2.2. Restricting the bisimulations of LTSs from Definition 2.1.6 to precharts, we obtain the following.

**Definition 2.2.5.** A *bisimulation* between precharts $(X, \delta_X)$ and $(Y, \delta_Y)$ is a relation $R \subseteq X \times Y$ such that if $(x, y) \in R$,

1. $x \xrightarrow{a} \checkmark$ if and only if $y \xrightarrow{a} \checkmark$,
2. if $x \xrightarrow{a} x'$, then there is a $y' \in Y$ such that $(x', y') \in R$ and $y \xrightarrow{a} y'$, and
3. if $y \xrightarrow{a} y'$, then there is an $x' \in X$ such that $x \xrightarrow{a} x'$ and $(x', y') \in R$.

If $(x, y)$ appears in a bisimulation, we say $x$ and $y$ are *bisimilar* and write $x \leftrightarrow y$. A bisimulation that is also an equivalence relation is a *bisimulation equivalence*.

Bisimulations are precisely the relations that preserve behaviour. Precharts can be organized in a category with *functional bisimulations* for homomorphisms, bisimulations that are the graphs of functions. The composition of two functional bisimulations is also a functional bisimulation, and the identity map is trivially a functional bisimulation. This is an important observation that follows from more general theoretical observations, which we cover briefly next.

### 2.2.1 Precharts are Coalgebras

So far, we have two different notions of bisimulation floating around—one for LTSs, another for precharts—and there are soon to be more. While the prechart-specific

case is of most interest to us in this chapter, there is an encompassing theory of behavioural equivalences between precharts, LTSs, and whatever other kinds of automata and transition systems might appear. One appropriate choice of framework for studying these objects is *(universal) coalgebra*, a category theoretic take on stateful systems. In coalgebra, system types are captured with endofunctors on the category **Set** of sets and functions [Gum99], as well as on other categories (see, for example, [Rut00; Mil10; Bon+12; Adá05; Jac16]).

**Definition 2.2.6.** Given an endofunctor $B$ on **Set**, called the *(coalgebraic) signature*, a *$B$-coalgebra* is a pair $(X, \delta_X)$ consisting of a set $X$ of *states* and a *structure map* $\delta_X \colon X \to BX$. A *$B$-coalgebra homomorphism* $h \colon (X, \delta_X) \to (Y, \delta_Y)$ is a function $h \colon X \to Y$ such that $\delta_Y \circ h = B(h) \circ \delta_X$, where $\circ$ denotes function composition. The category of $B$-coalgebras and their homomorphisms is written $\mathrm{Coalg}(G)$.

Precharts and their homomorphisms fit neatly into the framework of coalgebra. Let $\mathcal{P}_\omega X = \{U \subseteq X \mid U \text{ is finite}\}$. Given a set $X$, a subset $U \subseteq X$, and a function $f \colon X \to Y$, define

$$\begin{aligned} LX &= \mathcal{P}_\omega(Act \times (\checkmark + X)) \\ L(f)(U) &= \{(a, \checkmark) \mid (a, \checkmark) \in U\} \cup \{(a, f(x)) \mid (a, x) \in U\} \end{aligned} \tag{2.4}$$

Then $L$ is an endofunctor on **Set**, and precharts are precisely $L$-coalgebras. We use the terms "$L$-coalgebra" and "prechart" interchangeably.

### Charts and Subcoalgebras

Chart interpretations of star expressions also have a coalgebraic description: Given $r \in StExp$, $\langle r \rangle$ is the *smallest subcoalgebra* of $StExp$ containing $r$, i.e., if $U \subseteq StExp$ contains $r$ and $(U, \delta_U)$ is an $L$-coalgebra such that

$$\begin{array}{ccc} U & \xhookrightarrow{\mathrm{in}_U} & StExp \\ {\scriptstyle \delta_U}\downarrow & & \downarrow{\scriptstyle \ell} \\ PU & \xhookrightarrow{L(\mathrm{in}_U)} & L(StExp) \end{array} \tag{2.5}$$

commutes, then $\langle r \rangle \subseteq U$. In coalgebraic terminology, (2.5) states that the set $U$ is a *subcoalgebra* of $(StExp, \ell)$, meaning that it carries an $L$-coalgebra structure such that the inclusion of $U$ into $StExp$ is an $L$-coalgebra homomorphism $(U, \delta_U) \hookrightarrow (StExp, \ell)$.

Because $\mathrm{in}_U$ is injective, there is at most one $\delta_U$ such that $\mathrm{in}_U : (U, \delta_U) \hookrightarrow (StExp, \ell)$, so we sometimes refer to $U$ as a subcoalgebra of $(X, \delta)$.

### Homomorphisms and Bisimulations

We saw that functional bisimulations were the appropriate notion of behaviour-preserving maps between precharts. Functional bisimulations coincide with homomorphisms between $L$-coalgeberas.

**Lemma 2.2.7.** *A function $h : X \to Y$ between precharts $(X, \delta_X)$ and $(Y, \delta_Y)$ is an $L$-coalgebra homomorphism if and only if it is a functional bisimulation.*

Universal coalgebra also provides a general notion of bisimulation and functional bisimulation that makes sense for any $B$-coalgebras.

**Definition 2.2.8.** For a general endofunctor $B$, a *bisimulation* between two coalgebras $(X, \delta_X)$ and $(Y, \delta_Y)$ is a relation $R \subseteq X \times Y$ carrying a coalgebra structure $(R, \delta_R)$ such that the projection maps $\pi_1 : R \to X$ and $\pi_2 : R \to Y$ are $B$-coalgebra homomorphisms. If there is a bisimulation $R$ relating $x \in X$ and $y \in Y$, we say $x$ and $y$ are *bisimilar* and write $x \leftrightarrows_B y$. We often omit the signature and write $\leftrightarrows$ instead of $\leftrightarrows_B$ when it is clear from context.

Setting $B = L$, Definition 2.2.8 becomes Definition 2.2.5. We can also instantiate Definition 2.2.8 to bisimulations between LTSs (Definition 2.1.6) by setting

$$\begin{aligned} TX &= \mathcal{P}(\checkmark + Act \times X) \\ T(f)(U) &= \{\checkmark \mid \checkmark \in U\} \cup \{(a, f(x)) \mid (a,x) \in U\} \end{aligned} \tag{2.6}$$

and noticing that $T$-coalgebras and LTSs coincide.

Furthermore, any union of bisimulations is also a bisimulation. This follows from general facts about $B$-coalgebras, which I record here for future use.

**Lemma 2.2.9.** *Let $B$ be any endofunctor on* **Set**.

1. *If $\{(X_i, \delta_i)\}_{i \in I}$ is any set of coalgebras, there is a unique coalgebra structure on the disjoint union $\bigsqcup_{i \in I} X_i$ that defines their coproduct in* $\mathrm{Coalg}(B)$.
2. *If $h : (X, \delta_X) \to (Y, \delta_Y)$ is a coalgebra homomorphism, then there is a unique coalgebra structure $h(\delta)$ on its image $h[X]$ such that $h : (X, \delta_X) \to (h[X], h(\delta))$ and $(h[X], h(\delta)) \hookrightarrow (Y, \delta_Y)$.*

Proofs of Lemma 2.2.9 appear in many places [Rut00; Gum99; Jac16; Adá05][3]. I often make use of Lemma 2.2.9's constructions, so I have reproduced them below.

*Constructions.* Given a disjoint union of sets $\bigsqcup_{i\in I} X_i$ with inclusion maps $\mathrm{in}_i\colon X_i \hookrightarrow \bigsqcup_{i\in I} X_i$, there is a canonical map

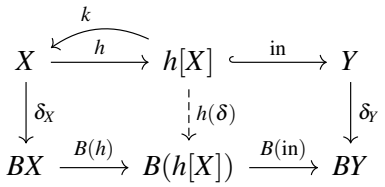
$$[\mathrm{in}_i^B]_{i\in I}\colon \bigsqcup_{i\in I} BX_i \to B\left(\bigsqcup_{i\in I} X_i\right) \qquad \mathrm{in}_i^B(\xi) = B(\mathrm{in}_i)(\xi) \text{ if } \xi \in BX_i \tag{2.7}$$

This is used in the coproduct construction: Given a set of $B$-coalgebras $\{(X_i,\delta_i)\}_{i\in I}$, their coproduct can be obtained as $\sum_{i\in I}(X_i,\delta_i) = (\bigsqcup_{i\in I} X_i, \sum_{i\in I}\delta_i)$, where

$$\sum_{i\in I}\delta_i = [\mathrm{in}_i^B]_{i\in I}\circ\bigsqcup_{i\in I}\delta_i\colon \bigsqcup_{i\in I} X_i \to \bigsqcup_{i\in I} BX_i \to B\left(\bigsqcup_{i\in I} X_i\right)$$

For the image construction, given $h\colon (X,\delta_X)\to(Y,\delta_Y)$, consider its factorization $h = \mathrm{in}\circ h'\colon X \twoheadrightarrow h[X] \hookrightarrow Y$. Let $k\colon h[X]\to X$ *split* $h'$, i.e., $h'\circ k = \mathrm{id}_{h[X]}$. Then the map $h(\delta) = B(h')\circ\delta_X\circ k$ is a coalgebra structure on $h[X]$. Since

$$\begin{aligned} B(\mathrm{in})\circ h(\delta) &= B(\mathrm{in})\circ B(h')\circ\delta_X\circ k \\ &= B(\mathrm{in}\circ h')\circ\delta_X\circ k \\ &= B(h)\circ\delta_X\circ k \\ &= \delta_Y\circ h\circ k \\ &= \delta_Y\circ\mathrm{in}\circ h'\circ k \\ &= \delta_Y\circ\mathrm{in} \end{aligned}$$




we have $\mathrm{in}\colon (h[X],h(\delta))\to(Y,\delta_Y)$. On the other hand,

$$B(\mathrm{in})\circ h(\delta)\circ h' = \delta_Y\circ\mathrm{in}\circ h' = \delta_Y\circ h = B(h)\circ\delta_X = B(\mathrm{in})\circ B(h')\circ\delta_X$$

If $Y=\emptyset$, then $X=\emptyset$ and $h'=\mathrm{id}$. Otherwise, $B(\mathrm{in})$ is injective (because it splits), so $h'\colon (X,\delta_X)\to(h[X],h(\delta))$ as desired. □

In fact, the last construction can be carried out when there exists only a coalgebra $(X,\delta_X)$ and a surjection $q\colon X\twoheadrightarrow Y$ such that $q(x)=q(y)$ implies $B(q)\circ\delta_X(x) = B(q)\circ$

[3] Actually, many of these sources prove something that subsumes Lemma 2.2.9, that the forgetful functor $(X,\delta)\mapsto X$ *creates colimits*.

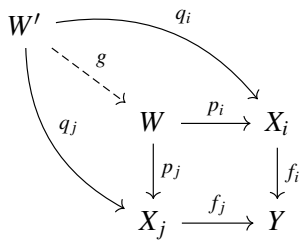
$\delta_X(y)$ for any $x, y \in X$. There exists a unique coalgebra structure $(Y, \delta_Y)$ such that $q\colon (X, \delta_X) \to (Y, \delta_Y)$, defined $\delta_Y = B(q) \circ \delta_X \circ k$ where $k$ is any right inverse of $q$ ($\delta_Y$ does not depend on $k$).

Finally, to see that the union of a set $\{(R_i, \delta_i)\}_{i \in I}$ of bisimulations between coalgebras $(X, \delta_X)$ and $(Y, \delta_Y)$ is itself a bisimulation, construct their coproduct $\sum_{i \in I}(R_i, \delta_i)$ and take the image of this coalgebra under the union of the inclusion maps $\bigsqcup_{i \in I} R_i \to X \times Y$. This has the following immediate consequence.

**Lemma 2.2.10.** *Between any two $B$-coalgebras, the relation $\underline{\leftrightarrow}$ is a bisimulation. Within a single $B$-coalgebra $(X, \delta_X)$, $\underline{\leftrightarrow}$ is a bisimulation.*

We return to general bisimilarity in Chapter 4. For now, in the concrete cases of LTSs and precharts, the functors $L$ and $T$ share a property that makes working with their versions of bisimilarity much easier. Namely, they *preserve weak pullbacks* [Gum98].

**Definition 2.2.11.** Given a set of arrows $D = \{f_i\colon X_i \to Y\}_{i \in I}$, a *cone* of $D$ is a set of arrows $\{p_i\colon W \to X_i\}_{i \in I}$, such that for any $i, j \in I$, $f_i \circ p_i = f_j \circ p_j$. A cone $\{p_i\colon W \to X_i\}_{i \in I}$ of $D$ is called a *weak pullback* of $D$ if every cone $\{q_i\colon W' \to X_i\}_{i \in I}$ admits an arrow $g\colon W' \to W$ such that $p_i \circ g = q_i$ for all $i \in I$.




An endofunctor $B$ *preserves* weak pullbacks if for finite $I$, $\{B(p_i)\}_{i \in I}$ is a weak pullback of $\{B(f_i)\}$ whenever $\{p_i\colon W \to X_i\}_{i \in I}$ is a weak pullback of $\{f_i\colon X_i \to Y\}_{i \in I}$.

Given a function $f\colon X \to Y$, define the *kernel* of $f$ to be

$$\ker(f) := \{(x, y) \mid f(x) = f(y)\}$$

Also, given relations $R \subseteq X \times Y$ and $Q \subseteq Y \times Z$, define their *composition* to be

$$R\,;Q = \{(x, z) \mid (\exists y \in Y)\ R(x, y) \text{ and } Q(y, z)\}$$

**Lemma 2.2.12** (Rutten [Rut00])**.** *Let $B$ be a weak pullback preserving endofunctor.*

1. *If $R \subseteq X \times Y$ and $Q \subseteq Y \times Z$ are bisimulations between $(X, \delta_X)$, $(Y, \delta_Y)$, and $(Z, \delta_Z)$, then so is $R;Q = \{(x,z) \mid (\exists y \in Y)\ xRyRz\}$.*
2. *On a fixed $B$-coalgebra, $\leftrightarrows$ is a bisimulation equivalence,*
3. *A relation $R \subseteq X \times X$ on $(X, \delta_X)$ is a bisimulation equivalence if and only if it is the kernel of a coalgebra homomorphism with domain $(X, \delta_X)$.*
4. *A map $h\colon (X, \delta_X) \to (Y, \delta_Y)$ is a $B$-coalgebra homomorphism if and only if the graph $\mathrm{Gr}(h)$ of $h$ is a bisimulation relation.*

Observe that item 4 of Lemma 2.2.12 is Lemma 2.2.7 with $B = L$, and provides us an analogous characterization of LTS homomorphisms in the case of $B = T$.

*Remark* 2.2.13. Another property of $B$-coalgebras that is implied by weak pullback preservation is the existence of $\langle x \rangle_{\delta_X}$ for each state $x$ of a finite $B$-coalgebra $(X, \delta_X)$. If $B$ preserves weak pullbacks, then for any finite collection $\{U_i \mid i \leq n\}$ of subcoalgebras of $(X, \delta_X)$, $\bigcap U_i$ is also a subcoalgebra [Rut00]. Thus, $\langle x \rangle_{\delta_X}$ is obtained as the (necessarily finite) intersection of all subcoalgebras of $(X, \delta_X)$ containing $x$.

### 2.2.2 Axiomatizing Bisimilarity

Within the prechart $(StExp, \ell)$, bisimilarity captures a number of intuitive equivalences, keeping in mind the interpretation of star expressions as processes. For instance, $0r \leftrightarrows 0$ and $r + s \leftrightarrows s + r$ for any $r, s \in StExp$. These are two of the axioms suggested by Milner in [Mil84], appearing as (B7) and (B2) in Figure 2.1.

**Definition 2.2.14.** Let 1fMil be the inference system consisting of (B0)-(B3), (B6)-(B8), (FP1), and the rule

$$\frac{s = r_1 s + r_2}{s = r_1 * r_2} \tag{RSP}$$

We write $r \equiv_* s$ to denote that $1\mathsf{fMil} \vdash r = s$ and say *$r$ and $s$ are provably equivalent*.

*Remark* 2.2.15. In [GF20], Grabmayer and Fokkink include an additional axiom, (BKS2), which states that $(r*s)t = r*(st)$. This is, in fact, derivable from the smaller set of axioms:

$$1\mathsf{fMil} \vdash (r*s)t \overset{(FP1)}{=} (r(r*s)+s)t \overset{(B8)}{=} (r(r*s))t + st \overset{(B7)}{=} r((r*s)t) + st \overset{(RSP)}{=} r*(st)$$

The following theorem, due to Grabmayer and Fokkink [GF20], implies that 1fMil defines a sound axiomatization of bisimilarity in $(StExp, \ell)$.

**Theorem 2.2.16** (Soundness)**.** *The relation $\equiv_* \subseteq StExp \times StExp$ is a bisimulation equivalence in $(StExp, \ell)$.*

*Proof.* We show that the relation $\equiv_*$ satisfies the conditions of a bisimulation (Definition 2.2.5) by induction on the proof of the statement $1\mathsf{fMil} \vdash r = s$. The base case is mostly routine, and covers the equational axioms and reflexivity. We show the case for the identity (B7) $r_1(r_2r_3) = (r_1r_2)r_3$, as it is slightly more involved.

Let $r_1, r_2, r_3 \in StExp$. We need to show that $r_1(r_2r_3) \xrightarrow{a} \checkmark$ if and only if $(r_1r_2)r_3 \xrightarrow{a} \checkmark$, and that $r_1(r_2r_3) \xrightarrow{a} s$ implies there is a transition $(r_1r_2)r_3 \xrightarrow{a} s'$ with $s \leftrightarrows s'$ and $(r_1r_2)r_3 \xrightarrow{a} s'$ implies there is a transition $r_1(r_2r_3) \xrightarrow{a} s$ with $s \leftrightarrows s'$.

Given any $a \in Act$, $s \in StExp$, $r_1(r_2r_3) \xrightarrow{a} s$ implies either (i) $r_1 \xrightarrow{a} \checkmark$ and $s = r_2r_3$, or (ii) $r_1 \xrightarrow{a} r'$ and $s = r'(r_1r_2)$. In case (i), $r_1r_2 \xrightarrow{a} r_2$ using the transition rule for sequential composition. It follows that $(r_1r_2)r_3 \xrightarrow{a} r_2r_3$. In case (ii), $r_1r_2 \xrightarrow{a} r'r_2$, so $(r_1r_2)r_3 \xrightarrow{a} (r'r_2)r_3$. We have both $r_2r_3 \equiv_* r_2r_3$ and $r_1(r_2r_3) \equiv_* (r_1r_2)r_3$ as desired.

Conversely, $(r_1r_2)r_3 \xrightarrow{a} s$ implies $r_1r_2 \xrightarrow{a} s'$ and $s = s'r_3$ for some $s' \in StExp$, because $\neg(r_1r_2 \xrightarrow{a} \checkmark)$. Therefore, either (i) $r_1 \xrightarrow{a} \checkmark$ and $s' = r_2$, or (ii) $r_1 \xrightarrow{a} r'$ and $s' = r'r_2$. In case (i), $r_1(r_2r_3) \xrightarrow{a} r_2r_3$. In case (ii), $r_1(r_2r_3) \xrightarrow{a} r'(r_2r_3)$. We have both $r_2r_3 \equiv_* r_2r_3, r1(r_2r_3) \equiv_* (r_1r_2)r_3$ as desired.

The more involved inductive cases are (RSP) and the congruence rule for $*$.

- Suppose the last step in the proof is

$$\frac{s = r_1s + r_2}{s = r_1 * r_2} \tag{RSP}$$

and the pair $(s, r_1s + r_2)$ satisfies Definition 2.2.5. If $s \xrightarrow{a} \checkmark$, then $r_1s + r_2 \xrightarrow{a} \checkmark$, which must mean that $r_2 \xrightarrow{a} \checkmark$ because $\neg(r_1s \xrightarrow{a} \checkmark)$. In such a case, $r_1 * r_2 \xrightarrow{a} \checkmark$. Conversely, if $r_1 * r_2 \xrightarrow{a} \checkmark$, then $r_2 \xrightarrow{a} \checkmark$ and therefore $r_1s + r_2 \xrightarrow{a} \checkmark$. By the induction hypothesis, $s \xrightarrow{a} \checkmark$. Hence, $s \xrightarrow{a} \checkmark$ iff $r_1 * r_2 \xrightarrow{a} \checkmark$.
If $s \xrightarrow{a} s' \neq \checkmark$, then there is an $r'$ such that $r_1s + r_2 \xrightarrow{a} r'$ and $s' \equiv_* r'$. If $r_2 \xrightarrow{a} r'$, then $r_1 * r_2 \xrightarrow{a} r'$. Else, if $r_1s \xrightarrow{a} r'$, then either $r_1 \xrightarrow{a} r''$ and $r' = r''s$, or $r_1 \xrightarrow{a} \checkmark$ and $r' = s$. In the former case, since $s \equiv_* r_1 * r_2$, $r_1 * r_2 \xrightarrow{a} r''(r_1 * r_2) \equiv_* r'$ because

$r' = r''s$. In the latter case, $r_1 * r_2 \xrightarrow{a} s = r'$.

Conversely, if $r_1 * r_2 \xrightarrow{a} r'$, then either $r' = r''(r_1 * r_2)$ where $r_1 \xrightarrow{a} r''$, or $r_2 \xrightarrow{a} r'$. In the former case, $r_1 s + r_2 \xrightarrow{a} r's$. By the induction hypothesis, there is a transition $s \xrightarrow{a} s'$ such that $s' \equiv_* r's \equiv_* r'(r_1 * r_2)$. In the latter case, by the induction hypothesis applied to $s \equiv_* r_1 s + r_2$, there is a transition $s \xrightarrow{a} s'$ such that $s' \equiv_* r'$. This concludes the (RSP) step.

- For the $*$ congruence rule, let $r_1 \equiv_* s_1$ and $r_2 \equiv_* s_2$ satisfy the conditions of Definition 2.2.5. Then

$$\begin{aligned} r_1 * s_1 \xrightarrow{a} \checkmark &\iff s_1 \xrightarrow{a} \checkmark \\ &\iff s_2 \xrightarrow{a} \checkmark && \text{(ind. hyp.)} \\ &\iff r_2 * s_2 \xrightarrow{a} \checkmark \end{aligned}$$

Also, $r_1 \xrightarrow{a} \checkmark$ if and only if $r_2 \xrightarrow{a} \checkmark$ by the induction hypothesis, so $r_1 * s_1 \xrightarrow{a} r_1 * s_1$ if and only if $r_2 * s_2 \xrightarrow{a} r_2 * s_2$ as well. Finally, suppose $r_1 \xrightarrow{a} r'$. Then $r_1 * s_1 \xrightarrow{a} r'(r_1 * s_1)$, and $r_2 \xrightarrow{a} r''$ for some $r'' \equiv_* r'$ by the induction hypothesis. It follows that, $r_2 * s_2 \xrightarrow{a} r''(r_2 * s_2) \equiv_* r'(r_1 * s_1)$. Conversely, if $s_1 \xrightarrow{a} r'$, then $r_1 * s_1 \xrightarrow{a} r'$. By the induction hypothesis, there is a transition $s_2 \xrightarrow{a} r'' \equiv_* r'$ and therefore $r_2 * s_2 \xrightarrow{a} r'' \equiv_* r'$. This concludes the $*$ congruence case. □

The role that bisimulation equivalences play in coalgebra is analogous to the role that congruences play in algebra. By Theorem 2.2.16 and Lemma 2.2.12, the set $StExp/\equiv_*$ of star expressions modulo provable equivalence is itself an $L$-coalgebra, and the quotient map $[-]_{\equiv_*} : StExp \to StExp/\equiv_*$ is a coalgebra homomorphism (see Lemma 2.2.9 item 2).

### 2.2.3 Left-affine Systems and Solutions

Starting with an expression $r \in StExp$, obtaining a prechart $(X, \delta_X)$ with a state $x \in X$ such that $r \leftrightarrows x$ is only a matter of computing $\langle r \rangle$. Going from a prechart $(X, \delta_X)$ and a state $x \in X$ to an expression $r \in StExp$ such that $r \leftrightarrows x$ is more difficult (and not always possible). Next, we discuss how to see star expressions modulo bisimilarity as solutions to certain systems of equations obtained from labelled transition systems.

Given a prechart $(X, \delta_X)$, its *left-affine system* is the set of formal equations

$$x = \sum_{x \xrightarrow{a} \checkmark} a + \sum_{x \xrightarrow{a} x'} ax' \tag{2.8}$$

indexed by $X$, where we are thinking of each $x \in X$ as an indeterminate. A *solution* to the left-affine system associated with $X$ is a map $\varphi \colon X \to StExp$ such that

$$\varphi(x) \equiv_* \sum_{x \xrightarrow{a} \checkmark} a + \sum_{x \xrightarrow{a} x'} a\ \varphi(x') \tag{2.9}$$

for all $x \in X$. Composing a solution $\varphi$ with the homomorphism $[-]_{\equiv_*} \colon StExp \to StExp/{\equiv_*}$, (2.9) becomes the equation

$$[\varphi(x)]_{\equiv_*} = \sum_{x \xrightarrow{a} \checkmark} a + \sum_{x \xrightarrow{a} x'} a\ [\varphi(x')]_{\equiv_*} \tag{2.10}$$

It follows from (2.10) that $R = \{(x, [\varphi(x)]_{\equiv_*}) \mid x \in X\}$ is a bisimulation between $(X, \delta)$ and $StExp/{\equiv_*}$. Since $R$ is the graph of the map $[-]_{\equiv_*} \circ \varphi$, if $\varphi \colon X \to StExp$ is a solution, then $[-]_{\equiv_*} \circ \varphi \colon X \to StExp/{\equiv_*}$ is a coalgebra homomorphism. Conversely, if $[-]_{\equiv_*} \circ \varphi$ is a homomorphism, then (2.10) holds. Since (2.9) and (2.10) are equivalent, we have proven the following result.

**Lemma 2.2.17.** *Let $(X, \delta)$ be a prechart. A map $\varphi \colon X \to StExp$ is a solution to $(X, \delta)$ iff the map $[-]_{\equiv_*} \circ \varphi \colon X \to StExp/{\equiv_*}$ is a homomorphism $(X, \delta) \to (StExp/{\equiv_*}, [\ell]_{\equiv_*})$.*

The following theorem tells us that the quotient map $[-]_{\equiv_*}$ is a solution to the left-affine system of $(StExp, \ell)$.

**Theorem 2.2.18** (Fundamental)**.** *Let $r \in StExp$. Then*

$$r \equiv_* \sum_{r \xrightarrow{a} \checkmark} a + \sum_{r \xrightarrow{a} s} as$$

*where $\sum_{i=1}^{n} r_i = r_1 + (\sum_{i=2}^{n} r_i)$ is a generalized sum that is well-defined up to the commutativity and associativity of $+$ assumed in Figure 2.1.*

*Remark* 2.2.19. The above theorem is originally due to Brzozowski [Brz64]. It is named the *fundamental theorem of regular expressions* by Silva [Sil10], in analogy

with the fundamental theorem of formal power series [Rut03] (a version of the fundamental theorem of calculus).

In the following pages, we often identify solutions with their corresponding homomorphisms into $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$. This is justified by Lemma 2.2.17.

## 2.3 A Local Approach

In the previous section, we observed that 1fMil is sound with respect to bisimilarity, and that *solutions* from [GF20] coincide with coalgebra homomorphisms into $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$. In op. cit., Grabmayer and Fokkink show that 1fMil is *complete* with respect to bisimilarity: that $r \equiv_* s$ whenever $r \leftrightarrows s$, for any $r, s \in StExp$. Next, we give an abstract description of Grabmayer and Fokkink's approach to proving soundness and completeness, which we call the *local approach*, and compare it to an approach found in classical automata theory. Grabmayer and Fokkink's approach can essentially be organized into four steps.

**Step 1** is to show that $\equiv_*$ is a bisimulation equivalence. This is the content of Theorem 2.2.16 from Section 2.2, and establishes soundness.

**Step 2** is to identify a class $\mathbf{C}$ of precharts such that for any $r \in StExp$, $\langle r \rangle \in \mathbf{C}$.

**Step 3** is to show that for any $(X, \delta) \in \mathbf{C}$, there is a unique $L$-coalgebra homomorphism $(X, \delta) \to (StExp/{\equiv_*}, [\ell]_{\equiv_*})$. By Lemma 2.2.17, homomorphisms into $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$ are identifiable with solutions, so this is the same as saying that precharts in $\mathbf{C}$ admit unique solutions.

**Step 4** is to show that $\mathbf{C}$ is closed under binary coproducts and *bisimulation collapses*. That is, for any $(X, \delta_X), (Y, \delta_Y) \in \mathbf{C}$, we also find $(X \sqcup Y, \delta_X \sqcup \delta_Y) \in \mathbf{C}$ as well as the *bisimulation collapse* of $(X, \delta_X)$, $(X/{\leftrightarrows}, [\delta_X]_{\leftrightarrows}) \in \mathbf{C}$, obtained from the quotient-by-bisimilarity map $[-]_{\leftrightarrows}$ (see Lemma 2.2.9).

Thus, the four steps above are a coalgebraic rephrasing of their approach that requires the introduction of coproducts. However, the coalgebraic analogue of Grabmayer and Fokkink's distinguished class of models is easily seen to be closed under binary coproducts, as we will see in Section 2.4.

*Remark* 2.3.1. It should be noted that Grabmayer and Fokkink never explicitly show in [GF20] that their class $\mathbf{C}$, of what we call well-layered precharts (ignoring

start states), is closed under binary coproducts. However, it is readily seen from the definition that a coproduct of layering witnesses is a layering witness for the coproduct of their underlying precharts. Thus, the class of well-layered precharts is closed under binary coproducts.

The four steps above are sufficient for showing soundness and completeness of an axiomatization of bisimilarity in general. In fact, we can even replace **Step 4** with

**Step 4** is to show that $\mathbf{C}$ is *collapsible*, i.e.,for any $(X,\delta_X),(Y,\delta_Y) \in \mathbf{C}$ and any $x \in X$ and $y \in Y$ such that $x \leftrightarrows y$, there is a $(V,\delta_V) \in \mathbf{C}$ and a pair of homomorphisms $p\colon (X,\delta_X) \to (V,\delta_V)$ and $q\colon (Y,\delta_Y) \to (V,\delta_V)$ such that $p(x) = q(y)$.

Steps 1-4 constitute what we call the *local approach*, leading to soundness and completeness via the following theorem.

**Theorem 2.3.2.** *Let $B$ be a weak pullback preserving endofunctor and fix a $B$-coalgebra $(E,\varepsilon)$. Furthermore, let*

- ***(Step 1)*** *$\equiv$ be a bisimulation equivalence on $(E,\varepsilon)$ and*
- ***(Step 2)*** *$\mathbf{C}$ be a class of $B$-coalgebras containing $\langle e\rangle$ for each $e \in E$,*

*such that*

- ***(Step 3)*** *there is exactly one homomorphism $(X,\delta_X) \to (E/{\equiv},[\varepsilon]_\equiv)$ for every $B$-coalgebra $(X,\delta_X) \in \mathbf{C}$, and*
- ***(Step 4)*** *$\mathbf{C}$ is collapsible.*

*Then $e \equiv f$ if and only if $e \leftrightarrows f$ for any $e,f \in E$.*

*Proof.* Let $e,f \in E$. Since $\equiv$ is a bisimulation equivalence, $e \equiv f$ implies $e \leftrightarrows f$ for any $e,f \in E$ by definition. Therefore, it suffices to show the converse.

Suppose $e \leftrightarrows f$, and let $(X,\delta_X) = \langle e\rangle$ and $(Y,\delta_Y) = \langle f\rangle$. As $\mathbf{C}$ is collapsible, there is a $(V,\delta_V) \in \mathbf{C}$ and a pair of homomorphisms $p\colon (X,\delta_X) \to (V,\delta_V), q\colon (Y,\delta_Y) \to (V,\delta_V)$ such that $p(e) = q(f)$. Let $\varphi\colon V \to E/{\equiv}$ be a homomorphism and consider

$$\begin{array}{ccccc} X & \xrightarrow{\ p\ } & V & \xleftarrow{\ q\ } & Y \\ \big\downarrow & & \big\downarrow{\scriptstyle \varphi} & & \big\downarrow \\ E & \xrightarrow{[-]_\equiv} & E/{\equiv} & \xleftarrow{[-]_\equiv} & E \end{array}$$

Since $(X, \delta_X)$ and $(Y, \delta_Y)$ admit unique homomorphisms into $(E/{\equiv}, [\varepsilon]_\equiv)$, this diagram commutes. In particular, $[e]_\equiv = \varphi(p(e)) = \varphi(q(f)) = [f]_\equiv$, meaning $e \equiv f$. □

A class of $B$-coalgebras that is closed under binary coproducts and bisimulation collapses is collapsible: If $(X, \delta_X)$ and $(Y, \delta_Y)$ are in the class, and $x \leftrightarrows y$ for some $x \in X$ and $y \in Y$, let $V = (X \sqcup Y)/{\leftrightarrows}$ and take $p = [-]_{\leftrightarrows} \circ \mathrm{in}_X$ and $q = [-]_{\leftrightarrows} \circ \mathrm{in}_Y$. Here, $\mathrm{in}_X : X \hookrightarrow X \sqcup Y$ is the inclusion of $X$ into the coproduct $X \sqcup Y$, and similarly for $\mathrm{in}_Y$, and $[-]_{\leftrightarrows} : X \sqcup Y \twoheadrightarrow V$ is the bisimulation collapse homomorphism. Because $x \leftrightarrows y$ in $(X, \delta_X) + (Y, \delta_Y)$, $p(x) = q(y)$, from which collapsibility follows.

**Steps 1** through **3** of the local approach might be familiar to readers acquainted with completeness proofs in classical automata theory. Salomaa's proof of completeness for the whole axiom set in Figure 2.1 with respect to language equivalence for regular expressions can be reorganized as follows: Kleene proved in [Kle56] that a language is regular if and only if it is recognized by a state in a deterministic finite automaton (or DFA). Thus, we can take DFAs as the distinguished class of coalgebras in **Step 2**. This trivializes **Step 4**, as finiteness is preserved under binary coproducts and bisimulation collapses. The central difficulty in Salomaa's completeness proof is in **Step 3** [Sal66]. Kozen's proof of completeness for Kleene algebra [Koz91] can be reorganized similarly.

Although all four steps had been taken, neither of the completeness proofs in [Sal66; Koz91] end by taking quotients or bisimulation collapses (like in Theorem 2.3.2). Instead, bisimulations between DFAs are treated as DFAs, and solutions are pulled back across projection homomorphisms. As Grabmayer and Fokkink point out in [GF20], this use of bisimulations does not translate to the case of one-free regular expressions. This is due to the fact that the distinguished class **C**, consisting of the precharts for which they could prove the existence and uniqueness of solutions, does not include every bisimulation between precharts in **C**. This is where the need for collapsibility becomes apparent.

Comparing the difficulties in Salomaa's approach with the difficulties in Grabmayer and Fokkink's approach reveals a crucial aspect of discovering soundness and completeness theorems in general. When choosing a distinguished class of models **C**, there is a balance to be found between the difficulty of finding solutions to models in **C** and proving their uniqueness on the one hand, and ensuring desirable structural

qualities of **C** on the other. Salomaa circumvented the difficulties of **Steps 2** and **4** by including every finite automaton in his distinguished class, but this made **Step 3** a more difficult problem. Grabmayer and Fokkink were able to take **Step 3** and prove uniqueness of solutions for precharts in their distinguished class with relative ease, but **Step 4** took great ingenuity.

## 2.4 Layered Loop Existence and Elimination

Grabmayer and Fokkink prove that Milner's axioms are complete with respect to bisimilarity for the one-free fragment by modelling star expressions with charts. They single out a specific class of charts, namely those satisfying their *layered loop existence and elimination property*, or *LLEE-property* for short. Roughly, a prechart is said to satisfy the LLEE-property if there is a labelling of its transitions by natural numbers such that an edge descending into a loop accompanies a descent in natural number labellings, and such that no successful termination can occur mid-loop. The existence of such a labelling ensures that loops are never mutually nested, and requires threads to finish every task in a loop before termination. Every chart interpretation of a star expression has the LLEE-property, and every prechart with the LLEE-property admits a unique solution.

In this section, we discuss a coalgebraic version of Grabmayer and Fokkink's distinguished class of models, the class of so-called *LLEE-precharts*, and review the proof of its collapsibility. As it so happens, a slight variation of Grabmayer and Fokkink's proof of collapsibility shows something much stronger: That the class of finite LLEE-precharts is closed under arbitrary homomorphic images. The main tool used in the proof of collapsibility is the *connect-through-to* operation, which preserves bisimilarity while it identifies bisimilar states. We generalize Grabmayer and Fokkink's connect-through-to operation, and show that it can be used to establish closure under homomorphic images in general.

### 2.4.1 Well-layeredness

We give an equivalent but different characterization of LLEE-precharts that makes them easier to describe coalgebraically, and rename the property to *well-layeredness*. While we recall the necessary details, much of what is covered here can be found in more detail in [GF20].

A simple but interesting observation about well-layeredness is that it makes no reference to the action labels of a prechart. In other words, well-layeredness is really a property of *transition systems with termination* (*transition systems* for short), coalgebras for the endofunctor $\mathcal{P}_\omega(\checkmark + \mathrm{Id})$. A well-layered transition system is a transition system that admits a particular labelling, called an *entry/body labelling*, that satisfies a few extra conditions.

**Definition 2.4.1.** For any set $X$, define the transformation

$$\begin{aligned} \mathsf{und}_X &: \mathcal{P}_\omega(\checkmark + \{\mathsf{e},\mathsf{b}\} \times X) \longrightarrow \mathcal{P}_\omega(\checkmark + X) \\ \mathsf{und}_X(U) &= \{\checkmark \mid \checkmark \in U\} \cup \{x \mid (\mathsf{e},x) \in U\} \cup \{x \mid (\mathsf{b},x) \in U\} \end{aligned} \tag{2.11}$$

An *entry/body labelling* of a transition system $(X,\tau)$ is a coalgebra $(X,\tau^\bullet)$ for the endofunctor $\mathcal{P}_\omega(\checkmark + \{\mathsf{e},\mathsf{b}\} \times \mathrm{Id})$ such that $\mathsf{und}_X \circ \tau^\bullet = \tau$. We use the notation $\mathsf{und}_*(X,\tau^\bullet) = (X, \mathsf{und}_X \circ \tau^\bullet)$.

To state the extra conditions on the labellings that define well-layeredness, we need some notation. Given an entry/body labelling $(X,\tau^\bullet)$, the following glyphs are used to denote its various transition types: For any $x,y \in X$,

- $x \to \checkmark$ means $\checkmark \in \tau^\bullet(x)$,
- $x \to_{\mathsf{e}} y$ means $(\mathsf{e},y) \in \tau^\bullet(x)$,
- $x \to_{\mathsf{b}} y$ means $(\mathsf{b},y) \in \tau^\bullet(x)$,
- $x \curvearrowright y$ means there exists $v_1,\ldots,v_k$ such that $x \notin \{v_1,\ldots,v_k,y\}$ and

$$x \to_{\mathsf{e}} v_1 \to_{\mathsf{b}} \cdots \to_{\mathsf{b}} v_k \to_{\mathsf{b}} y$$

- $x \circlearrowright y$ means there exists $v_1,\ldots,v_k$ such that $x \notin \{v_1,\ldots,v_k\}$ and $y \in \{v_1,\ldots,v_k\}$

$$x \to_{\mathsf{e}} v_1 \to_{\mathsf{b}} \cdots \to_{\mathsf{b}} v_k \to_{\mathsf{b}} x$$

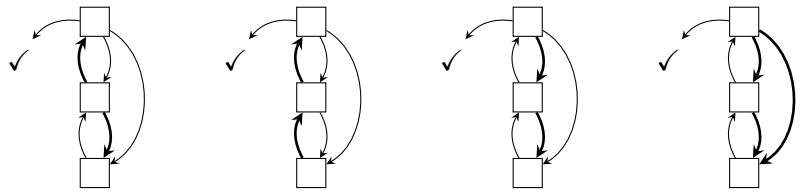
Transitions of the form $x \to_{\mathsf{e}} y$ and $x \to_{\mathsf{b}} y$ are called *entry* and *body* transitions, respectively. We enclose a relation in $(-)^+$ or $(-)^*$ to denote its transitive or transitive-reflexive closure respectively.

**Definition 2.4.2.** A *layering witness* is an entry/body labelling $(X,\tau^\bullet)$ that is

1. *locally finite*, meaning that $\langle x \rangle$ is finite for all $x \in X$,
2. *flat*, meaning that $x \to_{\mathsf{e}} y$ implies $\neg(x \to_{\mathsf{b}} y)$ for all $x, y \in X$,
3. *fully specified*, meaning that for all $x, y \in X$,
   (a) $\neg(x \to_{\mathsf{b}}^{+} x)$, and
   (b) if $x \to_{\mathsf{e}} y$ for some $y \neq x$, then $y \to^{+} x$.
4. *layered*, meaning that the directed graph $(X, \curvearrowright)$ is acyclic, and
5. *goto-free*, meaning that $x \curvearrowright y$ implies $\neg(y \to \checkmark)$, for all $x, y \in X$.

A transition system $(X, \delta)$ is said to be *well-layered* if $(X, \delta) = \mathsf{und}_*(X, \tau^\bullet)$ for a layering witness $(X, \tau^\bullet)$.

*Example* 2.4.3. The first three out of the following entry/body labellings are not well-layered, but the fourth is.



Here we use bold arrows to denote $\to_{\mathsf{e}}$-transitions and ordinary arrows for $\to_{\mathsf{b}}$-transitions. The first and second violate the goto-free condition (5). The second additionally violates layeredness (4), and the third is not fully-specified (3a).

We will also use $\mathsf{und}$ to denote the transformation

$$\begin{aligned} \mathsf{und}_X &\colon LX \to \mathcal{P}_\omega(\checkmark + X) \\ \mathsf{und}_X(U) &= \{\checkmark \mid \checkmark \in U\} \cup \bigcup_{a \in Act} \{x \mid (a, x) \in U\} \end{aligned} \tag{2.12}$$

Again, we define $\mathsf{und}_*(X, \delta_X) = (X, \mathsf{und}_X \circ \delta_X)$ and call it the *underlying transition system* of $(X, \delta_X)$. A layering witness *for a prechart* is a layering witness for its underlying transition system, and a prechart is said to be *well-layered* if it has a layering witness.

*Remark* 2.4.4. It can be checked that the underlying transition system of a locally finite prechart $(X, \delta_X)$ is well-layered if and only if $(X, \delta_X)$ has an *LLEE-witness* [GF20].

To obtain an LLEE-witness from a layering witness, replace each $x \to_{\mathsf{e}} y$ with a

weighted transition $x \xrightarrow{[|x|_{en}]} y$, where[4]

$$|x|_{en} = \max\{m \in \mathbb{N} \mid (\exists x_1, \ldots, x_m)\ x \curvearrowright x_1 \curvearrowright \cdots \curvearrowright x_m \text{ and } x \neq x_i \neq x_j \text{ for } i \neq j\}$$

and each $x \to_{\mathsf{b}} y$ with $x \xrightarrow{[0]} y$. This is a well-defined translation because we have assumed that $\langle x \rangle$ is finite and $(\langle x \rangle, \curvearrowright)$ is acyclic.

To obtain a layering witness from an LLEE-witness, replace each $x \xrightarrow{a}_{[n]} y$ by $x \to_{\mathsf{e}} y$ if $n > 0$ and $y \to^+ x$, or by $x \to_{\mathsf{b}} y$ otherwise. This entry/body labelling is flat because every resulting entry transition appears in a minimal cycle, and every minimal cycle contains precisely one entry transition by (W1) and (W2)(b) from [GF20]. Each of the remaining conditions are by construction, or are a direct consequence of the LLEE-witness conditions. For example, full specification follows from local finiteness and (W1) in op. cit., and our assumption that $x \to_{\mathsf{e}} y$ implies $y \to^+ x$ for all $x, y$.

By restricting a layering witness $(X, \delta^\bullet)$ to a subcoalgebra $(U, \delta_U)$ of $(X, \delta_X) = \mathsf{und}_*(X, \delta^\bullet)$, one obtains a layering witness $(U, \delta_U^\bullet)$ for $(U, \delta_U)$. It follows from this observation and the lemma below that $\langle r \rangle$ is well-layered for any $r \in \mathit{StExp}$.

**Lemma 2.4.5.** *The prechart $(\mathit{StExp}, \ell)$ is well-layered.*

*Proof.* To see that $(\mathit{StExp}, \ell)$ is locally finite, one can adapt Antimirov's proof of finiteness for LTSs generated by regular expressions [Ant96] to see that $\langle r \rangle$ is finite for each $r \in \mathit{StExp}$. For the rest, we construct a layering witness $(\mathit{StExp}, \ell^\bullet)$ as follows: Label a transition $r \xrightarrow{a} s$ with $r \to_{\mathsf{e}} s$ if $r \to_{\mathsf{e}} s$ can be derived from the rules below.

$$\frac{r_1 \to_{\mathsf{e}} s}{r_1 r_2 \to_{\mathsf{e}} s r_2} \qquad \frac{r_1 \xrightarrow{a} \checkmark}{r_1 * r_2 \to_{\mathsf{e}} r_1 * r_2} \qquad \frac{r_1 \xrightarrow{a} s \qquad r_1 \to^+ \checkmark}{r_1 * r_2 \to_{\mathsf{e}} s(r_1 * r_2)}$$

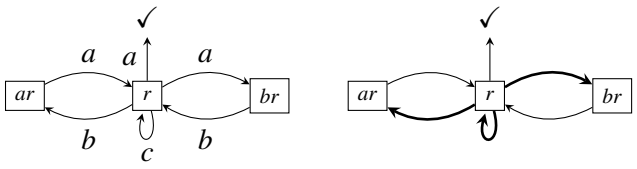
Label all other transitions as $\to_{\mathsf{b}}$-transitions. It is easy to see this labelling is flat.

One can derive that $(\mathit{StExp}, \ell^\bullet)$ is fully specified, layered, and goto-free from [GF20, Proposition 3.7] and the observation in Remark 2.4.4, that each layering witness corresponds to an LLEE-witness and vice versa. □

[4] Note here that either $x = y$, or the displayed equation includes $y$ as a candidate $x_1$, i.e., $x \curvearrowright y$. Indeed, if a prechart has an LLEE-witness, then it has an LLEE-witness that is *flat*, meaning that if $x \xrightarrow{[n]} y$ and $x \xrightarrow{[m]} y'$ for $n, m > 0$, then $n = m$. Intuitively, the number $|x|_{en}$ measures the depth of the deepest nested loop in a star expression representing $x$.

This completes Step 2 from Section 2.3: Where $\mathbf{C}$ is the set of finite well-layered precharts, we find $\langle r \rangle \in \mathbf{C}$ for any $r \in StExp$.

*Example* 2.4.6. Let $s = ab + ba + c$ and $r = s * a$, where $a, b, c \in A$. The prechart $\langle r \rangle$ is depicted below along with a layering witness.




*Remark* 2.4.7. It is important to note that not every well-layered prechart has a unique layering witness. The prechart $\langle (aa) * 0 \rangle$, for example, has exactly two.

## 2.4.2 A note about natural transformations in coalgebra

Before we continue with the next steps of the completeness proof, I would like to briefly mention the involvement of the transformations named und from a theoretical perspective. The key point is that und is an example of a natural transformation.

Given a natural transformation $\eta : B_1 \Rightarrow B_2$ between endofunctors $B_1, B_2$ on **Set**, we obtain a functor $\eta_* : \mathrm{Coalg}(B_1) \to \mathrm{Coalg}(B_2)$, defined $\eta_*(X, \delta) = (X, \eta_X \circ \delta)$, with many nice properties [Rut00], including the following.

**Lemma 2.4.8.** *Let $\eta : B_1 \Rightarrow B_2$ for endofunctors $B_1$ and $B_2$ on* **Set**.

1. *If $U \subseteq X$ is a subcoalgebra of $(X, \delta_X)$, then $U$ is a subcoalgebra of $\eta_*(X, \delta_X)$.*
2. *If $R \subseteq X \times Y$ is a bisimulation between $(X, \delta_X)$ and $(Y, \delta_Y)$, then $R$ is a bisimulation between $\eta_*(X, \delta_X)$ and $\eta_*(Y, \delta_Y)$. In particular, for $x \in X$ and $y \in Y$, if $x \leftrightarrows_{B_1} y$, then $x \leftrightarrows_{B_2} y$.*
3. *If $h : (X, \delta_X) \to (Y, \delta_Y)$ is a $B_1$-coalgebra homomorphism, then $h$ is also a $B_2$-coalgebra homomorphism $\eta_*(X, \delta_X) \to \eta_*(Y, \delta_Y)$.*

Concretely, every bisimulation $(R, \delta_R)$ between precharts $(X, \delta)$ and $(Y, \delta)$ carries an underlying bisimulation $\mathsf{und}_*(R, \delta_R)$ between the transition systems $\mathsf{und}_*(X, \delta_X)$ and $\mathsf{und}_*(Y, \delta_Y)$. However, not every bisimulation between $\mathsf{und}_*(X, \delta_X)$ and $\mathsf{und}_*(Y, \delta_Y)$ lifts to a bisimulation between $(X, \delta)$ and $(Y, \delta)$: Such relations ignore action labels in general, while bisimulations between precharts do not.

### 2.4.3 Existence and uniqueness of solutions

Steps 1 and 2 consisted of showing that $\equiv_*$ is a bisimulation equivalence and $\langle r \rangle$ is a well-layered prechart for each $r \in \mathit{StExp}$. To complete step 3 of the local approach, Grabmayer and Fokkink give an explicit description of a solution to a chart $(X, \delta) = \langle x \rangle$ with layering witness $(X, \delta^\bullet)$, called the *canonical solution*, and show that it is equivalent to any other solution to $(X, \delta)$. For any $x \in X$, let

$$\varphi_X(x) \equiv_* \left( \sum_{x \xrightarrow{a}_{\mathsf{e}} x} a + \sum_{\substack{x \xrightarrow{a}_{\mathsf{e}} y \\ x \neq y}} a\, t_X(y,x) \right) * \left( \sum_{x \xrightarrow{a} \checkmark} a + \sum_{x \xrightarrow{a}_{\mathsf{b}} y} a\, \varphi_X(y) \right) \tag{2.13}$$

where

$$t_X(y,x) \equiv_* \left( \sum_{y \xrightarrow{a}_{\mathsf{e}} y} a + \sum_{\substack{y \xrightarrow{a}_{\mathsf{e}} z \\ z \neq y}} a\, t_X(z,y) \right) * \left( \sum_{y \xrightarrow{a}_{\mathsf{b}} x} a + \sum_{\substack{y \xrightarrow{a}_{\mathsf{b}} z \\ x \neq z}} a\, t_X(z,x) \right)$$

The canonical solution $\varphi_X(x)$ is defined recursively on the quantity

$$|x|_{bo} = \max\{m \mid (\exists x_1, \ldots, x_m)\ x \to_{\mathsf{b}} x_1 \to_{\mathsf{b}} \cdots \to_{\mathsf{b}} x_m\}$$

The expression $t_X(y,z)$ is defined when $x \curvearrowright y$, recursively on the pair $(|x|_{en}, |y|_{bo})$ in the lexicographical ordering on $\mathbb{N} \times \mathbb{N}$, where $|x|_{en}$ is given in Remark 2.4.4.

It is shown in [GF20] that for any solution $\varphi\colon X \to \mathit{StExp}$, $\varphi(x) \equiv_* \varphi_X(x)$ for all $x \in X$. This proves that well-layered *charts* have unique solutions. The same result readily extends to the prechart case: If $(X, \delta)$ is an arbitrary well-layered prechart and $x \in X$, then $\varphi_{\langle x \rangle}(x)$ is a well-defined expression, as $\langle x \rangle$ is a subcoalgebra of $(X, \delta)$ and is therefore also well-layered. By uniqueness of solutions for charts, the map $\varphi_X : X \to \mathit{StExp}/{\equiv}$ given by $\varphi_X(x) = \varphi_{\langle x \rangle}(x)$ is a well-defined solution to $(X, \delta)$. Furthermore, since every solution to $(X, \delta)$ restricts to a solution to $\langle x \rangle$ for each $x \in X$, $\varphi_X$ is the unique solution to $(X, \delta)$.

**Lemma 2.4.9.** *If $(X, \delta)$ is a well-layered prechart with layering witness $(X, \delta^\bullet)$, then $\varphi_X : X \to \mathit{StExp}/{\equiv_*}$ is the unique solution to $(X, \delta)$*

*Proof.* That $\varphi_X$ is a solution is [GF20, Proposition 5.5]. That it is the unique solution

is [GF20, Proposition 5.8]. □

### 2.4.4 Reroutings and Closure under homomorphic images

The crucial step in Grabmayer and Fokkink's proof is step 4 of the local approach, showing that the bisimulation collapse of a finite well-layered chart is also well-layered. This is done in a step-by-step procedure that exhaustively identifies bisimilar states. In each step, a specially chosen pair $(w_1, w_2)$ of distinct bisimilar states is reduced to the singleton $w_2$ by *rerouting* all of $w_1$'s incoming transitions to $w_2$ and then deleting $w_1$.

Given a set $X$ and an element $z \in X$, write $X \setminus z$ for the set $\{x \in X \mid x \neq z\}$ when $z \in X$, $X \setminus U$ for the set $\{x \in X \mid x \notin U\}$ for $U \subseteq X$.

**Definition 2.4.10.** Given a pair $(x_1, x_2)$ of distinct states of a prechart $(X, \delta)$, the *connect-$x_1$-through-to-$x_2$ construction* returns the prechart $(X \setminus x_2, \delta[x_2/x_1])$, where

$$\delta[x_2/x_1](x)(a) = \begin{cases} \{(a, x_2)\} \cup (\delta(x) \setminus (a, x_1)) & (a, x_1) \in \delta(x) \\ \delta(x)(a) & \text{otherwise} \end{cases}$$

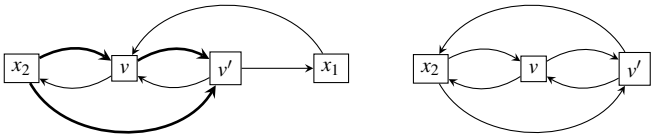
The connect-$x_1$-through-to-$x_2$ operation preserves bisimilarity, in the sense that if $R$ is a bisimulation equivalence on $(X, \delta)$, then $R \cap (X \times X \setminus x_1)$ is a bisimulation between $(X, \delta)$ and $(X \setminus x_1, \delta[x_2/x_1])$. This has the following consequence: If the only pairs of distinct states in $R$ are $(x_1, x_2)$ and $(x_2, x_1)$, then $R_1 = R \cap (X \times X \setminus x_1)$ is the graph of a homomorphism between $(X, \delta)$ and $(X \setminus x_1, \delta[x_2/x_1])$, and consequently $(X \setminus x_1, \delta[x_2/x_1]) \cong (X/R, [\delta]_R)$. Otherwise, $R|_{X \setminus x_1} = R \cap (X \setminus x_1)^2$ is a bisimulation equivalence containing a pair of distinct states $(x_3, x_4)$. If $(x_3, x_4)$ and $(x_4, x_3)$ are the only such pairs, then

$$R_2 = R|_{X \setminus x_1} \cap ((X \setminus x_1) \times (X \setminus \{x_1, x_3\}))$$

is the graph of a homomorphism $(X \setminus x_1, \delta[x_2/x_1]) \to (X \setminus \{x_1, x_3\}, \delta[x_2/x_1][x_4/x_3])$, so

$$R_1 ; R_2 = R \cap (X \times X \setminus \{x_1, x_3\})$$

is the graph of a homomorphism $(X, \delta) \to (X \setminus \{x_1, x_3\}, \delta[x_2/x_1][x_4/x_3])$ (where ; denotes relational composition) and $(X \setminus \{x_1, x_3\}, \delta[x_2/x_1][x_4/x_3]) \cong (X/R, [\delta]_R)$. Gen-




$(X, \delta^\bullet)$ $(X \setminus x_1, \delta[x_2/x_1])$

**Figure 2.5:** A bisimulation rerouting that does not preserve well-layeredness.

erally, if $X$ is finite, then iterating this construction yields the graph $R_1 ; \cdots ; R_m$ (for some $m$) of the homomorphism $(X, \delta) \to (X/R, [\delta]_R)$ (up to $\cong$). Taking $R = \leftrightarrows$, the bisimulation collapse of a finite prechart $X$ can be computed by iterating the connect-through-to operation until no distinct pairs of bisimilar states are left.

For an arbitrary well-layered prechart $(X, \delta)$ and a pair of distinct bisimilar states $(x_1, x_2)$, $(X \setminus x_1, \delta[x_2/x_1])$ may not be well-layered. An example discussed in [GF20] is the connect-through-to construction depicted in Figure 2.5, which takes a well-layered chart to a chart that does not admit a layering witness.

However, if $(x_1, x_2)$ is chosen carefully, then the connect-$x_1$-through-to-$x_2$ operation preserves well-layeredness. Where $(X, \delta^\bullet)$ is a layering witness for $(X, \delta)$, it is shown in [GF20] that $(X \setminus w_1, \delta[w_2/w_1])$ is well-layered for any pair $(w_1, w_2)$ of distinct bisimilar states satisfying one of the following three conditions in $(X, \delta^\bullet)$:

(C1) $\neg(w_2 \to^* w_1)$, and if $(\exists x)\ x \curvearrowright w_1$, then $\neg(\exists y)(w_2 \to^* y \Rightarrow)$

(C2) $w_2 \circlearrowleft^+ w_1$

(C3) $\neg(w_2 \to_{\mathsf{b}}^* w_1)$, and $(\exists x)\ w_1 \circlearrowleft x$ and $w_2 \circlearrowleft^+ x$ and if $w_1 \circlearrowleft y$, then $x \circlearrowleft y$

As Grabmayer and Fokkink point out [GF20], if $(X, \delta^\bullet)$ is a layering witness for a finite prechart $(X, \delta)$ such that $(X, \delta) \ncong (X/\leftrightarrows, [\delta]_{\leftrightarrows})$, then there is a pair $(w_1, w_2)$ of distinct bisimilar states satisfying one of (C1)-(C3) in $(X, \delta^\bullet)$. A slight variation on their proof yields the following.

**Lemma 2.4.11.** *Let $(X, \delta^\bullet)$ be a layering witness for $(X, \delta)$, and $R$ be a bisimulation equivalence on $(X, \delta)$. If $R$ is* non-trivial*, i.e., $(X, \delta) \ncong (X/R, [\delta]_R)$, then there is a pair $(w_1, w_2) \in R$ of distinct states satisfying one of (C1)-(C3).*

*Proof.* This is essentially [GF20, Proposition 6.4], where this is proven for $R = \leftrightarrows$. The proof of the above statement can be obtained by replacing $\leftrightarrows$ in their proof by an arbitrary bisimulation equivalence. □

By iterating the connect-through-to construction on the pairs guaranteed to exist in Lemma 2.4.11, every homomorphic image of a finite well-layered prechart is seen to be well-layered.

**Theorem 2.4.12.** *Let $(X,\delta)$ be a finite well-layered prechart, and $R$ be a bisimulation equivalence on $(X,\delta)$. Then $X/R$ is a well-layered prechart as well.*

This completes step 4 of Grabmayer and Fokkink's proof that Milner's axioms are complete with respect to bisimilarity for the one-free fragment of regular expressions.

**Theorem 2.4.13** ([GF20])**.** *For any $r,s \in StExp$, $r \leftrightarrows s$ if and only if $r \equiv_* s$.*

*Proof.* Let $\mathbf{C}$ be the set of finite well-layered precharts. By Lemma 2.4.5, $\langle r \rangle \in \mathbf{C}$ for any $r \in StExp$. Lemma 2.4.9 tells us that precharts in $\mathbf{C}$ admit unique solutions.

By Theorem 2.3.2, it suffices to show that $\mathbf{C}$ is collapsible. A class of coalgebras closed under binary coproducts and homomorphic images is collapsible, so by Theorem 2.4.12 it suffices to show that $\mathbf{C}$ is closed under binary coproducts. To this end, observe that if $(X,\delta_X^\bullet)$ and $(Y,\delta_Y^\bullet)$ are layering witnesses for $(X,\delta_X)$ and $(Y,\delta_Y)$ respectively, then $(X,\delta_X^\bullet)+(Y,\delta_Y^\bullet)$ is a layering witness for $(X,\delta_X)+(Y,\delta_Y)$. □

## 2.4.5 A note about reroutings in general

Interestingly, the connect-through-to construction can be performed on general $B$-coalgebras. For a fixed prechart $(X,\delta)$ and a pair of states $x_1,x_2 \in X$, if $i\colon X\setminus x_1 \hookrightarrow X$ is the inclusion map and $j\colon X \twoheadrightarrow X\setminus x_1$ is the map identifying $x_2$ with $x_1$, then the prechart $(X\setminus x_1, \delta[x_2/x_1])$ obtained from the connect-$x_1$-through-to-$x_2$ construction is given precisely by

$$\delta[x_2/x_1](x) = (\mathrm{id}_{Act} \times j)[\delta(x)] = L(j)\circ\delta\circ i(x)$$

In other words, the following diagram commutes.

$$\begin{array}{ccc} X\setminus x_1 & \xhookrightarrow{\ i\ } & X \\ {\scriptstyle \delta[x_2/x_1]}\downarrow & & \downarrow{\scriptstyle \delta} \\ L(X\setminus x_1) & \xleftarrow{\ L(j)\ } & LX \end{array}$$

Notice that $(i,j)$ is a *splitting*, meaning $j\circ i = \mathrm{id}_{X\setminus x_1}$.

**Definition 2.4.14.** Given any $B$-coalgebra $(X,\delta)$ and a splitting $(i,j)$ with $i\colon U \hookrightarrow X$, define $(X,\delta)[i,j] = (U, B(j)\circ\delta\circ i)$ and call $(X,\delta)[i,j]$ the *rerouting by* $(i,j)$ of $(X,\delta)$.

As is the case for the connect-through-to operation, reroutings that identify bisimilar states preserve bisimilarity.

**Lemma 2.4.15.** *Let $R$ be a bisimulation equivalence on a $B$-coalgebra $(X,\delta)$, and $(i,j)$ be a splitting with $i\colon U \hookrightarrow X$ and $\ker(j) \subseteq R$. Then $Q = R \cap (X\times U)$ is a bisimulation between $(X,\delta)$ and $(X,\delta)[i,j]$.*

*Proof.* Let $(R,\delta_R)$ be the coalgebra structure on $R$, and define $i_Q\colon Q \hookrightarrow R$ and $j_Q\colon R \twoheadrightarrow Q$ to be the maps $i_Q(x,y) = (x,i(y))$ and $j_Q(x,z) = (x,j(z))$. We need to check that $j_Q$ is, indeed, a map into $Q$. This follows from the observation that $j(z) = j\circ i\circ j(z)$, and therefore since $\ker(j)\subseteq R$, $(z, i\circ j(z)) \in R$. Because $R$ is transitive and both $(x,z),(z,i\circ j(z)) \in R$, $(x,i\circ j(z)) \in R$ as well. This means $j_Q(x,z) = (x,j(z)) \in Q$.

Define the coalgebra structure $(Q, \delta_Q[i_Q,j_Q])$, where

$$\delta_Q[i_Q,j_Q] = B(j_Q)\circ\delta_R\circ i_Q.$$

By definition, $i_Q$ and $j_Q$ satisfy $\pi_1^R\circ i_Q = \pi_1^Q$, $\pi_2^R\circ i_Q = i\circ\pi_2^Q$, and $\pi_2^R\circ j_Q = j\circ\pi_2^Q$. On the one hand, $\pi_1^Q\colon Q\to X$ is a coalgebra homomorphism by definition. On the other,

$$\begin{aligned}
\delta[i,j]\circ\pi_2^Q &= B(j)\circ\delta_X\circ i\circ\pi_2^Q && \text{(def. of } \delta[i,j]\text{)}\\
&= B(j)\circ\delta_X\circ\pi_2^R\circ i_Q && \text{(def. of } i_Q\text{)}\\
&= B(j)\circ B(\pi_2^R)\circ\delta_R\circ i_Q && (R \text{ is a bisim.})\\
&= B(j\circ\pi_2^R)\circ\delta_R\circ i_Q && (B \text{ is a functor})\\
&= B(\pi_2^Q\circ j_Q)\circ\delta_R\circ i_Q && \text{(def. of } j_Q\text{)}\\
&= B(\pi_2^Q)\circ B(j_Q)\circ\delta_R\circ i_Q && (B \text{ is a functor})\\
&= B(\pi_2^Q)\circ\delta[i_Q,j_Q]. && \text{(def. of } \delta[i_Q,j_Q]\text{)}
\end{aligned}$$

Thus, $Q$ is a bisimulation between $(X,\delta)$ and $(X,\delta)[i,j]$. □

A rerouting $(X,\delta)[i,j]$ is called an *$R$-rerouting* if $R$ is a bisimulation and $\ker(j)\subseteq R$. In case $R = \leftrightarrows$, we will use the phrase *bisimulation rerouting* instead.

Assuming $B$ weakly preserves pullbacks, the relational composition of two bisimulations is again a bisimulation (Lemma 2.2.12). In general, if $R$ is an equivalence on $X$, and $V \subseteq Y \subseteq X$, then $R \cap (X \times V) = (R \cap (X \times Y)) ; (R \cap (Y \times V))$ and $R \cap (Y \times Y)$ is an equivalence relation. Thus, by iterating Lemma 2.4.15, we obtain the following generalization of Theorem 2.4.12.

**Theorem 2.4.16.** *Let $B$ be an endofunctor that weakly preserves pullbacks, and $\mathbf{C}$ be a class of finite $B$-coalgebras closed under isomorphism. Then*

1. *If for any $(X, \delta) \in \mathbf{C}$ and any nontrivial bisimulation equivalence $R \subseteq X \times X$ there is a nontrivial $R$-rerouting $(U, \delta_U)$ of $(X, \delta)$ with $(U, \delta_U) \in \mathbf{C}$, then $\mathbf{C}$ is closed under homomorphic images.*
2. *If for any $(X, \delta) \in \mathbf{C}$ such that $X \ncong X/\leftrightarrows$ there is a nontrivial bisimulation rerouting $(U, \delta_U)$ of $(X, \delta)$ such that $(U, \delta_U) \in \mathbf{C}$, then $\mathbf{C}$ is closed under bisimulation collapses.*

As closure under bisimulation collapses is often enough to establish collapsibility, Theorem 2.4.16 tells us that establishing an abundance of reroutings in the distinguished class can be a crucial step towards completeness.

## 2.5 A Global Approach

We now discuss a different approach to proving soundness and completeness theorems in process algebra, which we call the *global approach*, and show how the soundness and completeness theorems of [GF20] fit in this setting. Fitting Grabmayer and Fokkink's proof into the mould of the global approach involves expanding the class of finite well-layered precharts to a much larger class that is closed under homomorphic images. We further show how the same remoulding technique can turn many local approach proofs into global ones.

The global approach originates in coalgebraic automata theory [Jac06; Sil10; SBR10; Mil10; BMS13]. Its main goal is to show that the expression language modulo provable equivalence is isomorphic to a subcoalgebra of a *final* coalgebra. For example, in [Jac06], Jacobs proves that the Kleene algebra axioms (see [Koz91; Con71]) are sound and complete with respect to language equivalence by exhibiting a coalgebra isomorphism between the initial Kleene algebra and the algebra of regular languages. The coalgebras that appear in Jacobs' paper are a standard choice for

deterministic automata, the $2 \times (-)^A$-coalgebras. This establishes the soundness and completeness of the Kleene Algebra axioms because bisimilarity and language equivalence coincide. Silva successfully applies the same method in [Sil10] to a variety of expression languages and axiomatizations parameterized by the functor $B$, with Jacobs' proof given by the special case $B = 2 \times (-)^A$. Following the same pattern, Milius gives an expression language and axiomatization of language equivalence for stream circuits in [Mil10], and generalizes some results in [Sil10] to endofunctors on categories other than **Set**. Following a similar approach, all three of the above are unified in [BMS13].

In order to explain precisely how the global approach works, fix a $B$-coalgebra $(E, \varepsilon)$, thought of as an abstract expression language, and let $\equiv$ be an equivalence relation on $E$. Similar to the local approach, the global approach involves a sequence of four steps:

**Step 1** is showing that $\equiv$ is a bisimulation equivalence. This establishes soundness.

**Step 2** consists of identifying a class $\mathbf{C}$ of $B$-coalgebras in which $(E/\equiv, [\varepsilon]_\equiv)$ is *weakly final* in $\mathbf{C}$, i.e., that $(E/\equiv, [\varepsilon]_\equiv) \in \mathbf{C}$ and every $(X, \delta) \in \mathbf{C}$ admits a homomorphism $(X, \delta) \to (E/\equiv, [\varepsilon]_\equiv)$. Again, homomorphisms into $(E/\equiv, [\varepsilon]_\equiv)$ play the role of solutions, so it can be said that coalgebras in $\mathbf{C}$ admit solutions.

**Step 3** is a proof that $(E/\equiv, [\varepsilon]_\equiv)$ is *subfinal* in $\mathbf{C}$, i.e., every $(X, \delta) \in \mathbf{C}$ admits at most one solution.

**Step 4** consists of showing that $\mathbf{C}$ is closed under homomorphic images.

These four steps are sufficient for showing the soundness and completeness of the axiomatizations in each of the cases considered in [Jac06; Sil10; SBR10; Mil10; BMS13] because the functors that are present there satisfy two key properties. The first key property is that there is a $B$-coalgebra $(Z, \zeta)$ that is *final*.

**Definition 2.5.1.** Let $B$ be an endofunctor and $\mathbf{C}$ be a class of $B$-coalgebras. A $B$-coalgebra $(Z, \zeta)$ is *final in* $\mathbf{C}$ if $(Z, \zeta) \in \mathbf{C}$ and for any $(X, \delta) \in \mathbf{C}$, there is a unique homomorphism $!_\delta \colon (X, \delta) \to (Z, \zeta)$. We call $(Z, \zeta)$ *final* if it is final in $\mathrm{Coalg}(B)$.

*Remark* 2.5.2. Whether a final $B$-coalgebra exists usually depends on set-theoretic properties of $B$ [AR94]. For example, while there is no final LTS [Lam68], there is a final prechart, due to the finite branching restriction implicit in (2.4).

Steps 1-4 above imply that $(E/\equiv, [\varepsilon]_\equiv)$ is a subcoalgebra of $(Z,\zeta)$.

**Lemma 2.5.3.** *Let $(E,\varepsilon)$ and $(Z,\zeta)$ be $B$-coalgebras with $(Z,\zeta)$ final. Then the behaviour map $!_\varepsilon\colon (E,\varepsilon)\to(Z,\zeta)$ is injective if and only if there is a class $\mathbf{C}$ of $B$-coalgebras that is closed under homomorphic images and such that $(E,\varepsilon)$ is final in $\mathbf{C}$.*

*Proof.* For the forward direction, let $\mathbf{C}$ be the class of $B$-coalgebras $(X,\delta)$ such that $!_\delta$ factors through $!_\varepsilon$. Note that the factorization of $!_\delta$ is necessarily unique because $!_\varepsilon$ is injective. Clearly, $(E,\varepsilon)$ is final in $\mathbf{C}$. To see closure under images, let $(X,\delta_X)\in\mathbf{C}$ and $h\colon (X,\delta_X)\to(Y,\delta_Y)$ be a surjective homomorphism. Let $!_{\delta_X} = !_\varepsilon \circ k$ be the factorization of $!_{\delta_X}$. Then there is a unique map $f\colon (Y,\delta_Y)\to(E,\varepsilon)$ such that $f(h(x)) = k(x)$, because

$$\ker(h)\subseteq\ker(!_\delta) = \ker(!_\varepsilon\circ k) = \ker(k)$$

To see that $f$ is a homomorphism, observe that

$$B(f)\circ\delta_Y\circ h = B(f)\circ B(h)\circ\delta_X = B(f\circ h)\circ\delta_X = B(k)\circ\delta_X = \varepsilon\circ k = \varepsilon\circ f\circ h$$

Since $h$ is surjective, this implies that $B(f)\circ\delta_Y = \varepsilon\circ f$.

For the other direction, let $J = !_\zeta[E]$ and use Lemma 2.2.9.*2* to build $\zeta_J = !_\varepsilon(\varepsilon)$. We know $(J,\zeta_J)\in\mathbf{C}$ from closure under homomorphic images. Since $(E,\varepsilon)$ is final in $\mathbf{C}$, $(J,\zeta_J)$ admits a unique coalgebra homomorphism $h\colon (J,\zeta_J)\to(E,\varepsilon)$. Composing, $h\circ !_\varepsilon\colon (E,\varepsilon)\to(E,\varepsilon)$ is a homomorphism, so finality of $(E,\varepsilon)$ in $\mathbf{C}$ tells us that $h\circ !_\varepsilon = \mathrm{id}_E$. As $!_\varepsilon$ has a left inverse, it is injective. □

This means that if every $(X,\delta)\in\mathbf{C}$ admits a unique solution and $\mathbf{C}$ is closed under homomorphic images, then $[e]_\equiv = !_\varepsilon(e)$ for any $e\in E$.[5] The second key property is preservation of weak pullbacks (Definition 2.2.11).

**Lemma 2.5.4** (Rutten [Rut00])**.** *Let $(X,\delta_X)$ and $(Y,\delta_Y)$ be $B$-coalgebras, $x\in X$, and $y\in Y$. Assume that a final $B$-coalgebra exists. If $B$ preserves weak pullbacks, then $x\leftrightarrows y$ if and only if $!_{\delta_X}(x) = !_{\delta_Y}(y)$.*

Following steps 1 through 4 above, and assuming that $B$ has a final coalgebra and preserves weak pullbacks, Lemma 2.5.3 and Lemma 2.5.4 tell us that $[e]_\equiv = !_\varepsilon(e) = !_\varepsilon(f) = [f]_\equiv$ if and only if $e\leftrightarrows f$, for any $e,f\in StExp$.

[5] Here, we have identified $(E/\equiv, [\varepsilon]_\equiv)$ with its isomorphic copy in $(Z,\zeta)$.

**Theorem 2.5.5.** *Assume $B$ preserves weak pullbacks and let $\equiv$ be a bisimulation equivalence on a $B$-coalgebra $(E,\varepsilon)$. Let $\mathbf{C}$ be a class of $B$-coalgebras that is closed under homomorphic images. If $(E/\equiv,[\varepsilon]_\equiv)$ is a final object in $\mathbf{C}$, then $e \equiv f$ if and only if $e \leftrightarrows f$ for any $e,f \in E$.*

*Remark* 2.5.6. Similar to Lemma 2.5.3, the converse of Theorem 2.5.5 is also true.

It follows from standard observations about the prechart functor $L$ that there is a final $L$-coalgebra [Rut00][6] and that $L$ preserves weak pullbacks [Gum99]. This suggests the possibility that the global approach can be taken to proving Theorem 2.2.16 and Theorem 2.4.13. This is indeed the case, although the class of finite well-layered precharts needs to be extended to include $(StExp/\equiv_*,[\ell]_{\equiv_*})$. We show how to do this in the next section.

### 2.5.1 A global approach to the one-free fragment

Returning to the one-free fragment of regular expressions, we have already seen that the class of well-layered precharts has $(StExp,\ell)$ as a member. It is likely that $(StExp/\equiv_*,[\ell]_{\equiv_*})$ is also well-layered, but this turns out to be unnecessary.

In order to have the *global* approach go through for the one-free fragment, we make a slight change in the distinguished class of precharts from Section 2.4.

**Definition 2.5.7.** Let $\mathbf{C}_{loc}$ be the class of *locally well-layered* precharts, i.e., $(X,\delta) \in \mathbf{C}_{loc}$ if and only if $(X,\delta)$ is locally finite and every $x \in X$ is contained in a well-layered subcoalgebra.

Every finite subcoalgebra of a locally well-layered prechart is well-layered. To see why, observe that a finite subcoalgebra of a locally well-layered prechart is a union of well-layered precharts. The union of a set of subcoalgebras is a homomorphic image of their coproduct, and we already know that the set of well-layered precharts is closed under disjoint unions and homomorphic images.

Using the fact that the finite well-layered precharts are closed under homomorphic images, we obtain the following key lemma.

**Lemma 2.5.8.** *Let $(X,\delta_X)$ be locally well-layered and $q\colon (X,\delta_X) \twoheadrightarrow (Y,\delta_Y)$ be a surjective coalgebra homomorphism. Then $(Y,\delta_Y)$ is locally well-layered as well.*

[6] Namely, that it is *bounded*.

*Proof.* Since $(X, \delta_X)$ is locally finite, and local finiteness is preserved under homomorphic images, $(Y, \delta_Y)$ is locally finite as well. Let $(U, \delta_U)$ be a finite subcoalgebra of $(Y, \delta_Y)$. By Theorem 2.4.12, it suffices to show that $(U, \delta_U)$ is the homomorphic image of a well-layered prechart.

To this end, let $U = \{y_1, \ldots, y_n\}$ and let $\{x_1, \ldots, x_n\} \subseteq X$ be such that $q(x_i) = y_i$ for $i = 1, \ldots n$. If $(V, \delta_V)$ is the smallest subcoalgebra of $(X, \delta_X)$ containing $\{x_1, \ldots, x_n\}$, then $q$ restricts to a subcoalgebra homomorphism on $V$, and by definition $(U, \delta_U)$ is the smallest subcoalgebra of $(Y, \delta_Y)$ containing $\{y_1, \ldots, y_n\}$. Whence, the restriction $q|_V \colon (V, \delta_V) \twoheadrightarrow (U, \delta_U)$ is a surjective homomorphism.

Since $\langle x_i \rangle$ is finite for each $i$, $V$ is finite and therefore $(V, \delta_V)$ is well-layered. Hence, $(U, \delta_U)$ is the homomorphic image of a well-layered finite prechart, and by Theorem 2.4.12 is well-layered. □

Every well-layered prechart is locally well-layered (a subcoalgebra of a well-layered prechart is well-layered), so $(StExp, \ell)$ is locally well-layered by Lemma 2.4.5. Since $(StExp, \ell) \in \mathbf{C}_{loc}$ and $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$ is the image of $StExp$ under the homomorphism $[-]_{\equiv} \colon (StExp, \ell) \to (StExp/{\equiv_*}, [\ell]_{\equiv_*})$, Lemma 2.5.8 tells us that $(StExp/{\equiv_*}, [\ell]_{\equiv_*}) \in \mathbf{C}_{loc}$ as well.

So far, we have taken step 4 and the first half of step 2 from the global approach. Interestingly, step 3 and the latter half of step 2 are possible because of Lemma 2.4.9, the uniqueness-of-solutions theorem for finite precharts. To see how this works, let $(X, \delta_X) \in \mathbf{C}_{loc}$. By Lemma 2.4.9, every finite subcoalgebra $U$ of $(X, \delta_X)$ admits a unique solution $\varphi_U \colon (U, \delta_U) \to (StExp/{\equiv_*}, [\ell]_{\equiv_*})$. Since homomorphisms restrict to subcoalgebras, this clearly implies that $(X, \delta_X)$ admits at most one solution. To see that $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$ is final in $\mathbf{C}_{loc}$, it suffices to construct a solution to $(X, \delta_X)$.

The unique solution to $(X, \delta_X)$ is the map $\varphi_X \colon (X, \delta_X) \to (StExp/{\equiv_*}, [\ell]_{\equiv_*})$ given by $\varphi_X(x) = \varphi_U(x)$ for any finite subcoalgebra $U$ of $(X, \delta_X)$ containing $x$. To see that this is well-defined, recall that $(X, \delta_X)$ is locally finite, meaning that every state of $X$ is contained in a finite subcoalgebra of $(X, \delta_X)$. If $(U, \delta_U)$ and $(V, \delta_V)$ are finite subcoalgebras of $(X, \delta_X)$ with $x \in U$ and $x \in V$, then $(U \cap V, \delta_{U \cap V})$ is a finite subcoalgebra of $(X, \delta_X)$ containing $x$. By closure under subcoalgebras, $(U \cap V, \delta_{U \cap V})$ is well-layered, so Lemma 2.4.9 tells us $(U \cap V, \delta_{U \cap V})$ admits a unique solution. Restricting $\varphi_U$ and $\varphi_V$ to $U \cap V$ also obtains a solution, so it must be that $\varphi_U(x) =$

$\varphi_{U\cap V}(x) = \varphi_V(x)$. To see that $\varphi$ is indeed a solution, observe that a map $h\colon X \to Y$ between locally finite coalgebras is a coalgebra homomorphism if $h|_U : U \to Y$ is a coalgebra homomorphism for any finite subcoalgebra $U$ of $(X,\delta_X)$. Since the latter statement is true of $\varphi$ by definition, $\varphi$ is a solution to $(X,\delta_X)$. This establishes the lemma below.

**Lemma 2.5.9.** *Let $\mathbf{C}_{loc}$ be the class of locally well-layered precharts. Then $(StExp/{\equiv_*}, [\ell]_{\equiv_*})$ is a final object in the class $\mathbf{C}_{loc}$.*

Together, Theorem 2.2.16 and Lemma 2.4.9, and Lemmas 2.5.8 and 2.5.9 constitute steps 1 through 4 of the global approach to proving soundness and completeness of Milner's axioms for the one-free fragment of regular expressions modulo bisimulation, thus providing an alternative proof of Theorem 2.4.13.

## 2.5.2 From local to global

As Lemma 2.5.9 illustrates, there are instances in which a completeness proof taking the global approach can be obtained from the four steps in the local approach. This is particularly the case when the distinguished class of coalgebras is closed under binary coproducts, like the well-layered precharts.

**Definition 2.5.10.** Given a class $\mathbf{C}$ of $B$-coalgebras, call a $B$-coalgebra $(X,\delta)$ *locally* $\mathbf{C}$ if for any $x \in X$, there is a subcoalgebra $(U,\delta_U) \in \mathbf{C}$ of $(X,\delta)$ such that $x \in U$. The class of locally $\mathbf{C}$ coalgebras is denoted $\mathbf{C}_{loc}$.

Where $(E,\varepsilon)$ is a locally finite $B$-coalgebra and $\equiv$ is a bisimulation equivalence on $(E,\varepsilon)$, assume that in the four steps of the local approach we have obtained a class $\mathbf{C}$ of finite $B$-coalgebras such that

(a) $(E,\varepsilon)$ is locally $\mathbf{C}$,
(b) each $(X,\delta_X) \in \mathbf{C}$ admits a unique homomorphism into $(E/{\equiv}, [\varepsilon]_\equiv)$, and
(c) $\mathbf{C}$ is closed under subcoalgebras, binary coproducts, and homomorphic images.

We argue that $\mathbf{C}_{loc}$ satisfies the necessary conditions for steps 2 through 4 of the global approach as follows.

To see uniqueness of solutions $(X,\delta_X) \to (E/{\equiv}, [\varepsilon]_\equiv)$ for $(X,\delta_X)$, we go through the same motions as in the prechart case. For any $(X,\delta_X) \in \mathbf{C}_{loc}$, the unique solution $\varphi_X : X \to E/{\equiv}$ is defined locally: If $x \in X$ and $(U,\delta_U) \in \mathbf{C}$ is a finite subcoalgebra of

$(X,\delta_X)$ containing $x$, then we define $\varphi_X(x) = \varphi_U(x)$, where $\varphi_U$ is the unique solution to $(U,\delta_U)$. The map $\varphi_X$ is a unique solution to $(X,\delta_X)$ for the same reason as in the prechart case: To see it is well-defined, suppose $(U,\delta_U),(V,\delta_V) \in \mathbf{C}$ are two subcoalgebras containing $x$. Then there is a subcoalgebra $W \subseteq U \cap V$ containing $x$. By closure under subcoalgebras, $(W,\delta_W) \in \mathbf{C}$, and $\varphi_U(x) = \varphi_W(x) = \varphi_V(x)$ by restriction.

To see closure under homomorphic images, let $h\colon (X,\delta_X) \twoheadrightarrow (Y,\delta_Y)$ be a surjective coalgebra homomorphism and $(X,\delta_X) \in \mathbf{C}_{loc}$. We need to show that for any $y \in Y$, there is a finite subcoalgebra $(U,\delta_U)$ containing $y$ that is in $\mathbf{C}$. To this end, find $x \in h^{-1}(y)$ and a finite subcoalgebra $(V,\delta_V) \in \mathbf{C}$ containing $x$. Let $(U,\delta_U)$ be the homomorphic image of $(V,\delta_V)$ under $h$. By closure under homomorphic images, $(U,\delta_U) \in \mathbf{C}$. Hence, $(Y,\delta_Y) \in \mathbf{C}_{loc}$.

Lastly, $(E,\varepsilon) \in \mathbf{C}_{loc}$ by assumption, so $(E/{\equiv},[\varepsilon]_\equiv) \in \mathbf{C}_{loc}$ by closure under homomorphic images. The following theorem obtains a global approach proof of completeness from the four steps of the local approach when $\mathbf{C}$ is closed under subcoalgebras, binary coproducts, and homomorphic images.

**Theorem 2.5.11.** *Let $\mathbf{C}$ be a class of finite B-coalgebras satisfying (a)-(c) above. Then $\mathbf{C}_{loc}$ is closed under homomorphic images, and $(E/{\equiv},[\varepsilon]_\equiv)$ is a final object of $\mathbf{C}_{loc}$.*

In other words, most coalgebraic completeness theorems proved using the local approach can be proved using the global approach. On the other hand, not every global approach-style completeness proof immediately gives rise to a local one. For example, few of the distinguished classes of coalgebras found in the global approach-style proofs in [Sil10] include the DFA interpretation of every expression in the language (these DFAs often fail to be locally finite).

## 2.6 Discussion

In this chapter, we saw a coalgebraic take on Grabmayer and Fokkink's completeness proof for Milner's axiomatization of bisimilarity of one-free star expressions [GF20]. We generalized their method to a completeness proof strategy called the local approach. In addition, we generalized a different completeness proof method, one that originates in [Jac06; Sil10; BMS13]. We called this generalized method the global approach. We saw that under mild conditions, a completeness proof taking the local approach can be remoulded to fit the global approach, and applied this remoulding

technique to Grabmayer and Fokkink's proof.

The crucial result that makes the global approach work is Lemma 2.5.3, which states that the behaviour map from a coalgebra $(E,\varepsilon)$ is injective if and only if $(E,\varepsilon)$ is final in some class of coalgebras closed under homomorphic images. At first glance, Lemma 2.5.3 is a rather elementary observation. Its proof is straightforward, and one always has on hand a candidate for the distinguished class (although it may not satisfy the unique solutions property): The set of quotients of $(E/\equiv,[\delta]_\equiv)$ itself is closed under homomorphic images. The significance of the lemma is not the depth of the result or in the difficulty of its proof. Instead, its significance is its direct and formal connection between properties that guarantee solvability of systems (i.e., the defining property of a class) and coalgebraic completeness theorems. It tells us that every coalgebraic completeness theorem can, in principle, be proven using the global approach.

Unfortunately, Lemma 2.5.3 does not provide a method of producing a completeness proof (that takes the global approach), even from an already established completeness theorem: It does not construct a distinguished class of coalgebras where solvability is easy to establish. In the next chapter, we prove several coalgebraic completeness theorems without using the local or global approaches introduced in this chapter. We use the global approach again in the last technical chapter (Chapter 4) to prove a general completeness theorem, for a family of process calculi (called effectful process calculi) where the distinguished class consists of the locally finite coalgebras.

## Chapter 3

# Guarded Kleene Algebra with Tests

In the previous chapter, we analyzed a completeness proof in the process algebra literature and drew from our analysis a general characterization of coalgebraic completeness theorems. In this chapter, we turn our attention to *guarded Kleene algebra with tests* (GKAT) and its bisimulation variant (bisimulation GKAT), two closely related algebraic structures for reasoning with imperative programs built from if-then-else and while loop constructs. We will also look at the *skip-free fragment* of (bisimulation) GKAT, which closely resembles the one-free regular expressions of the previous chapter. We are going to prove four completeness theorems: one for each of GKAT, bisimulation GKAT, skip-free GKAT, and skip-free bisimulation GKAT.

GKAT originates in *Kleene algebra with tests* (KAT), introduced by Kozen [KS96] for reasoning about semantics and equivalence of simple imperative programs. KAT extends Kleene Algebra [Kle56] with Boolean control flow to enable the encoding of conditionals and while loops. KAT has been applied to a number of verification tasks, including proof-carrying Java programs [KK05], compiler optimization [KP00], and file systems [Cha+19].

More recently, KAT was used for reasoning about packet-switched networks, serving as a core to NetKAT [And+14; Fos+15] and Probabilistic NetKAT [Fos+16; Smo+17]. The success of KAT in networking is partly due to its ability to both specify and verify network properties. Moreover, the implementations of NetKAT and ProbNetKAT were competitive with state-of-the-art tools [Fos+15; Smo+19]. This came as a surprise, because the decision problem for equivalence in both KAT and NetKAT is PSPACE-complete [KS96; Fos+15].

Further investigation [Smo+20] revealed that most NetKAT programs are deter-

ministic, and therefore make use of a proper fragment of KAT. It turns out that the difficulty of deciding equivalence in KAT can largely be attributed to nondeterminism in KAT programs. If one restricts to KAT programs that operate deterministically with respect to Boolean control flow, then the associated decision problem is *nearly linear*[1]. The deterministic fragment of KAT, in which programs are built up from if-then-else and while loop constructs, had already been introduced by Kozen and Tseng much earlier [KT08]. GKAT was introduced in [Smo+20] specifically for reasoning about deterministic KAT programs [Smo+20].

**Open problems**

Equivalence of GKAT programs was given a complete axiomatization in [Smo+20]. However, the axiomatization in op. cit. suffers from two serious drawbacks: First, it includes a powerful *uniqueness of solutions axiom*[2] (UA). Technically, UA is an axiom scheme that stands for infinitely many axioms, and greatly encumbers algebraic reasoning in practice. In order to use UA to show that two programs are equivalent, one needs to find a system of equations that they both satisfy. Second, the axiomatization contains an inference rule with a side condition reminiscent of Salomaa's *empty word property* for regular expressions [Sal66] (also see Figure 2.1). Kozen showed that Salomaa's empty word property is *non-algebraic* (it is not preserved under arbitrary substitution of atomic programs) [Koz91], which also impairs the use of axiomatic reasoning in context. The authors of [Smo+20] ask two questions in response to these drawbacks:

**Question 1.** Can UA be derived from the other GKAT axioms?

**Question 2.** Is there an algebraic axiomatization of GKAT?

Despite the attention GKAT has received recently [Sch+21; ZSS22; Sch+22; KSS23], both questions remain open.

**Contributions of this chapter**

The first contribution of this chapter is a new perspective on the semantics of GKAT programs and their corresponding automata. We show that the bisimilarity class of a

[1] In $\mathcal{O}(n\,\alpha(n))$, where $\alpha$ is the inverse Ackermann function [Tar75].

[2] The uniqueness axiom can also be seen as an instance of the Recursive Specification Principle from [BBK87].

GKAT program—a.k.a. its *behaviour*—is faithfully represented by a kind of tree. We also show that omitting a single axiom from the axiomatization of language equivalence for GKAT programs (presented in [Smo+20]) gives a complete axiomatization of bisimilarity of GKAT programs.

The second contribution is a partial answer to both **Question 1** and **Question 2** above. We show that axiomatically verifying an equivalence in GKAT does not require UA if the two programs in question are of a particular form: what we call *skip-free*. The fragment of GKAT consisting of skip-free programs is expressive enough to capture a large class of programs, and it also provides a better basis for algebraic reasoning than GKAT. In particular, we show that the non-algebraic side condition can be removed. Our inspiration to look at this fragment came from a completeness theorem of Grabmayer and Fokkink [GF20], whose proof we analyzed in Chapter 2.

In a nutshell, the contributions of this chapter are two-fold: One is to give a coalgebraic account of GKAT and its open completeness problems, and the other is to identify a large fragment of GKAT—the *skip-free fragment*—that admits an algebraic axiomatization. The chapter is organized as follows:

- We introduce GKAT in Section 3.1, starting with an example of the kind of algebraic reasoning it allows. We give an operational model of *bisimulation* GKAT, an axiomatization of bisimilarity for GKAT programs, and show that the bisimulation semantics of a GKAT program can be represented as a certain kind of tree. We state completeness theorems for both GKAT and bisimulation GKAT with the added axiom UA mentioned above. We do not prove the completeness of bisimulation GKAT with UA, as it is a special case of Theorem 4.5.12. We do prove the completeness of GKAT with UA: The proof is a reduction to the completeness of bisimulation GKAT using a so-called *pruning* technique.
- In Section 3.2, we introduce skip-free GKAT, the fragment of GKAT without nontrivial Boolean expressions, which we focus on for the remainder of the chapter. We introduce the operational models of skip-free GKAT programs, their language semantics, and their bisimulation semantics.
- We prove a completeness theorem for skip-free *bisimulation* GKAT in Section 3.3, an algebraic axiomatization of bisimilarity for skip-free GKAT programs. The completeness proof is a reduction to the completeness theorem of

Grabmayer and Fokkink [GF20] from Chapter 2.

- In Section 3.4, we prove that the algebraic axiomatization of skip-free GKAT is sound and complete with respect to the language semantics of GKAT using the completeness result for skip-free bisimulation GKAT and a pruning technique analogous to the one in Section 3.1. This provides a partial answer to both **Questions 1** and **2**.
- Much of the development of skip-free GKAT in this chapter is done without clarifying the connection to the original treatment of GKAT. The purpose of Section 3.5 is to make the connection explicit: We show that equivalence proofs of skip-free GKAT expressions (for both language and bisimulation semantics) embed into full GKAT. That is, (1) every skip-free (bisimulation) GKAT proof can be turned into a (bisimulation) GKAT proof with the same conclusion, and (2) every (bisimulation) GKAT proof that two skip-free expressions are equivalent can be turned into a skip-free (bisimulation) GKAT proof with the same conclusion.

Related work is discussed in Section 3.6.

This chapter is based on two papers.

Section 3.1 is based on *Guarded Kleene Algebra with Tests: Coalgebras, Coequations, and Completeness*, written by myself, Tobias Kappé, Dexter Kozen, and Alexandra Silva, and presented by myself at ICALP 2021 online [Sch+21].

The rest is based on *A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests*, written by Tobias Kappé, myself, and Alexandra Silva, and presented by myself at ESOP 2023 in Paris, France [KSS23].

## 3.1 Guarded Kleene Algebra with Tests

GKAT can be used to efficiently decide equivalences between propositional while programs, bits of code where control flow is fully specified and state changes in memory (like variable assignments) have been left uninterpreted. Reasoning about program equivalence in GKAT allows for an algebraic workflow consisting of certain code transformations. We start with an example of the kind of code transformations and algebraic reasoning possible in GKAT.

**Fizz! Buzz!**

In the game *Fizz! Buzz!* [Ree02], players sit in a circle taking turns counting up from one. Instead of saying any number that is a multiple of 3, players must say "fizz", and instead of a multiple of 5 players must say "buzz". If the number is a multiple of both 3 and 5, the player must say "fizz buzz".

```
def fizzbuzz1 =                                  (i)
  n := 1;
  while n ≤ 100 do
    if 3|n then
      if not 5|n then
        print fizz; n++;
      else
        print fizzbuzz; n++;
    else if 5|n then
      print buzz; n++;
    else
      print n; n++;
  print done!;
```

```
def fizzbuzz2 =                                  (ii)
  n := 1;
  while n ≤ 100 do
    if 5|n and 3|n then
      print fizzbuzz;
    else if 3|n then
      print fizz;
    else if 5|n then
      print buzz;
    else
      print n;
    n++;
  print done!;
```

**Figure 3.1:** Two possible specifications of the ideal *Fizz! Buzz!* player.

Imagine you are asked in a job interview to write a program that prints out the first 100 rounds of a perfect game of *Fizz! Buzz!*. You write the function fizzbuzz1 as given in Figure 3.1(i). Thinking about the interview later that day, you look up a solution, and you find fizzbuzz2, depicted in Figure 3.1(ii). You suspect that fizzbuzz2 should do the same thing as fizzbuzz1, and after thinking it over for a few minutes, you realize your program could be transformed into the reference solution by a series of transformations that do not change its semantics:

1. Place the common action $n{+}{+}$ at the end of the loop.
2. Replace not $5|n$ with $5|n$ and swap print *fizz* with print *fizzbuzz*.
3. Merge the nested branches of $3|n$ and $5|n$ into one.

Feeling somewhat more reassured, you ponder the three steps above. It seems like their validity is independent of the actual tests and actions performed by the code. For example, swapping the branches of an if-then-else block while negating the test should be valid under *any* circumstances. This raises a number of questions: Is there a family of primitive transformations that can be used to derive valid ways

of rearranging imperative programs? Furthermore, is there an algorithm to decide whether two programs are equivalent under these laws?

Guarded Kleene Algebra with Tests (GKAT) [Smo+20] has been proposed as a way of answering the questions above. Expressions in the language of GKAT model skeletons of imperative programs, where the exact meaning of tests and actions is abstracted. The syntax of GKAT appears in Section 3.1.1. GKAT programs can be interpreted essentially in two ways, as we will see in Section 3.1.2 and Section 3.1.3, and the interpretation you choose has a subtle influence on which programs are equivalent[3]. In Section 3.1.4, we see the laws of GKAT, which correspond to program transformations that are valid regardless of the semantics of tests and actions.

### 3.1.1 The Syntax

Formally, GKAT expressions are captured by a two-level grammar, generated by a finite set of tests $T$ and a finite set of actions $\Sigma$, as follows:

$$\begin{aligned} BExp \ni b, c &::= 0 \mid 1 \mid t \in T \mid b \vee c \mid b \wedge c \mid \overline{b} \\ GExp \ni e, f &::= b \in BExp \mid p \in \Sigma \mid e +_b f \mid ef \mid e^{(b)} \end{aligned} \tag{3.1}$$

*BExp* is the set of *Boolean expressions*, built from 0 (false), 1 (true), and primitive tests from $T$, and composed using $\vee$ (or), $\wedge$ (and), and $\overline{\ }$ (not). *GExp* is the set of GKAT *expressions*, built from tests (assert statements) and primitive actions $p \in \Sigma$. Here, $e +_b f$ is a condensed way of writing "if b then e else f", and $e^{(b)}$ is shorthand for "while b do e". The operator $\cdot$ models sequential composition.

*Example* 3.1.1. Abbreviating statements of the form print *foo* by simply writing *foo*, Figure 3.1(i) can be rendered as the GKAT expression

$$(n := 1) \cdot \left( \begin{array}{c} (fizz \cdot n{+}{+} +_{\overline{5|n}} fizzbuzz \cdot n{+}{+}) +_{3|n} \\ (buzz \cdot n{+}{+} +_{5|n} n \cdot n{+}{+}) \end{array} \right)^{(n \leq 100)} \cdot done! \tag{3.2}$$

Similarly, the program in Figure 3.1(ii) gives the GKAT expression

$$(n := 1) \cdot ((fizzbuzz +_{5|n \wedge 3|n} (fizz +_{3|n} (buzz +_{5|n} n))) \cdot n{+}{+})^{(n \leq 100)} \cdot done! \tag{3.3}$$

[3] You can rest assured that fizzbuzz1 and fizzbuzz2 are equivalent under both interpretations.

*Remark* 3.1.2. To avoid excessive use of parentheses in GKAT expressions, we use the order of operations ${}^{(b)} > \cdot > +_b$. We often omit $\cdot$ entirely. We also let each operation associate to the right. For example, $e_1e_2e_3 +_b e_4{}^{(b)}e_5 = (e_1 \cdot (e_2 \cdot e_3)) +_b (e_4{}^{(b)} \cdot e_5)$.

### 3.1.2 Relational/Language Semantics of GKAT

A moment ago, we stated that GKAT equivalences are intended to witness program equivalence, regardless of how primitive tests and actions are interpreted. We make this more precise by recalling the *relational* semantics of GKAT programs [Smo+20].[4] The intuition behind this semantics is that if the possible states of the machine being programmed are modelled by some set $S$, then tests are predicates on $S$ (comprised of all states where the test succeeds), and actions are relations on $S$ (encoding the changes in state affected by the action).

**Definition 3.1.3.** A *(relational) interpretation* is a triple $\sigma = (S, \mathsf{eval}, \mathsf{sat})$ where $S$ is a set, $\mathsf{eval} : \Sigma \to \mathcal{P}(S \times S)$ and $\mathsf{sat} : T \to \mathcal{P}(S)$. Each relational interpretation $\sigma$ gives rise to a semantics $[\![-]\!]_\sigma : \mathit{GExp} \to \mathcal{P}(S \times S)$, as follows:

$$
\begin{aligned}
[\![0]\!]_\sigma &= \emptyset & \left[\!\left[\bar{b}\right]\!\right]_\sigma &= [\![1]\!]_\sigma \setminus [\![b]\!]_\sigma \\
[\![1]\!]_\sigma &= \{(s,s) \mid s \in S\} & [\![p]\!]_\sigma &= \mathsf{eval}(p) \\
[\![t]\!]_\sigma &= \{(s,s) \mid s \in \mathsf{sat}(t)\} & [\![e +_b f]\!]_\sigma &= [\![b]\!]_\sigma \circ [\![e]\!]_\sigma \cup \left[\!\left[\bar{b}\right]\!\right]_\sigma \circ [\![f]\!]_\sigma \\
[\![b \wedge c]\!]_\sigma &= [\![b]\!]_\sigma \cap [\![c]\!]_\sigma & [\![ef]\!]_\sigma &= [\![e]\!]_\sigma \circ [\![f]\!]_\sigma \\
[\![b \vee c]\!]_\sigma &= [\![b]\!]_\sigma \cup [\![c]\!]_\sigma & \left[\!\left[e^{(b)}\right]\!\right]_\sigma &= ([\![b]\!]_\sigma \circ [\![e]\!]_\sigma)^* \circ \left[\!\left[\bar{b}\right]\!\right]_\sigma
\end{aligned}
$$

Here we use $\circ$ for relation composition and $^*$ for reflexive transitive closure.

*Remark* 3.1.4. If $\mathsf{eval}(p)$ is a partial function for every $p \in \Sigma$, then so is $[\![e]\!]_\sigma$ for each $e$. The above therefore also yields a semantics in terms of partial functions. Here, partiality is an inevitability when there is some test $b$ such that $\mathsf{sat}(b) \neq \Delta_S$, as $[\![bp]\!]_\sigma$ is the (domain) restriction of $\mathsf{eval}(p)$ to $\mathsf{sat}(b)$.

The relation $[\![e]\!]_\sigma$ contains the possible pairs of start and end states of the program $e$. For instance, the input-output relation of $[\![e +_b f]\!]$ consists of the pairs in $[\![e]\!]_\sigma$ (resp. $[\![f]\!]_\sigma$) where the start state satisfies $b$ (resp. violates $b$).

[4] A probabilistic semantics in terms of sub-Markov kernels is also possible [Smo+20].

*Example* 3.1.5. We can model the states of the machine running *Fizz! Buzz!* as pairs $(m,\ell)$, where $m$ is the current value of the counter $n$, and $\ell$ is a list of words printed so far. The accompanying maps sat and eval are given by:

$$\begin{aligned}
\mathsf{sat}(k|n) &= \{(m,\ell) \in S \mid m \equiv 0 \bmod k\} \\
\mathsf{sat}(n \leq k) &= \{(m,\ell) \in S \mid m \leq k\} \\
\mathsf{eval}(n{+}{+}) &= \{((m,\ell),(m+1,\ell)) \mid (m,\ell) \in S\} \\
\mathsf{eval}(n := k) &= \{((m,\ell),(k,\ell)) \mid (m,\ell) \in S\} \\
\mathsf{eval}(w) &= \{((m,\ell),(m,\ell w)) \mid (m,\ell) \in S\} \qquad (w \in \{\mathit{fizz}, \mathit{buzz}, \mathit{fizzbuzz}\}) \\
\mathsf{eval}(n) &= \{((m,\ell),(m,\ell m)) \mid (m,\ell) \in S\}
\end{aligned}$$

For instance, the interpretation of $n{+}{+}$ connects states of the form $(m,\ell)$ to states of the form $(m+1,\ell)$—incrementing the counter by one, and leaving the output unchanged. Similarly, print statements append the given string to the output.

On the one hand, this parameterized semantics shows that programs in the GKAT syntax can be given a semantics that corresponds to the intended meaning of their actions and tests. On the other hand, it allows us to quantify over all possible interpretations, and thus abstract from the meaning of the primitives.

As it happens, two expressions have the same relational semantics under any interpretation if and only if they have the same *language semantics* [Smo+20], i.e., they denote the same languages of *guarded strings*.

### Guarded Language Semantics

For a fixed set of primitive tests $T = \{t_1, \ldots, t_n\}$, let $At$ be the set of *atomic tests* formed from $T$, given by $At = \{t'_1 \wedge \cdots \wedge t'_n \mid t'_i \in \{t_i, \bar{t}_i\}\}$. We usually use $\alpha, \alpha', \beta$ to refer to atomic tests, often with subscripts. A *guarded string* is a word of alternating atomic tests and primitive actions that ends in an atomic test. Formally, the set of guarded strings is $GS = (At \cdot \Sigma)^* \cdot At$ (here, $\cdot$ denotes string concatenation). Intuitively, in a guarded string $\alpha_1 p_1 \alpha_2 p_2 \cdots p_n \alpha_{n+1}$, $\alpha_i$ captures the state of the program variables needed to execute the program action $p_i$, and the execution of each $p_i$ except the last yields a new program state $\alpha_{i+1}$. A *guarded language* is a set of guarded strings.

**Definition 3.1.6.** The *language semantics* $\mathcal{L}\colon GExp \to \mathcal{P}(GS)$ of GKAT is defined

$$\mathcal{L}(b) = b \qquad \mathcal{L}(p) = \{\alpha p \alpha' \mid \alpha, \alpha' \in At\} \qquad \mathcal{L}(e +_b f) = b \diamond \mathcal{L}(e) \cup \bar{b} \diamond \mathcal{L}(f)$$
$$\mathcal{L}(ef) = \mathcal{L}(e) \diamond \mathcal{L}(f) \qquad \mathcal{L}(e^{(b)}) = \bigcup_{n \in \mathbb{N}} (b \diamond \mathcal{L}(e))^{\diamond n} \diamond \bar{b}$$

where $L_1 \diamond L_2 = \{u\alpha w \mid u\alpha \in L_1 \text{ and } \alpha w \in L_2\}$, $L^{\diamond 0} = At$, and $L^{\diamond n+1} = L \diamond L^n$. If $\mathcal{L}(e) = \mathcal{L}(f)$, we say $e$ and $f$ are *language equivalent*.

*Remark* 3.1.7. Above, we identify $b \in BExp$ with the set $\{\alpha \in At \mid \alpha \leq_{\mathsf{BA}} b\}$ of atomic tests provably below $b$ in the Boolean algebra order (see Figure 3.3). Note that we could also have used the notation $\alpha \in b$ to denote that $\alpha \leq_{\mathsf{BA}} b$ for $\alpha \in At$, without ambiguity. This is justified by the finite version of Stone duality, that the Boolean algebra generated by $T$ (a finite set of tests) is isomorphic to the Boolean algebra of sets $2^{At}$. In fact, replacing $BExp$ with $2^{At}$ in the definition produces an algebra of programs that is equivalent to GKAT, but instead of having one if-then-else operation and one while loop operator for every Boolean expression, it has only $2^{At}$ of each—i.e., it has finitely many algebraic operations in total. To better fit the canonical literature, I have stuck to the original notation. See also Example 4.5.1.

It is shown in [Smo+20] that language equivalence for GKAT expressions is decidable in nearly linear time, and therefore so is equivalence under all relational interpretations. The decision procedure for language equivalence in [Smo+20] uses *bisimulation* and known results from automata theory, which are particularly good for mechanization. This is part of the motivation for the *bisimulation semantics* of GKAT, which interprets GKAT expressions as bisimilarity classes of states in a certain kind of automaton.

### 3.1.3 Bisimulation Semantics of GKAT

We can think of a GKAT program as a machine that evolves as it reads a string of atomic tests. Depending on the most recently observed atomic test, the program either accepts, rejects, or emits an action label and changes to a new state. For example, feeding "if $b$ do $p$ else $q$" an atomic test $\alpha \leq b$ causes it to perform the action $p$ and then terminate successfully.

$$\frac{\alpha \leq b}{b \Rightarrow \alpha} \qquad \frac{}{p \xrightarrow{\alpha|p} 1}$$

$$\frac{\alpha \leq b \qquad e \Rightarrow \alpha}{e +_b f \Rightarrow \alpha} \qquad \frac{\alpha \leq \bar{b} \qquad f \Rightarrow \alpha}{e +_b f \Rightarrow \alpha} \qquad \frac{\alpha \leq b \qquad e \xrightarrow{\alpha|p} e'}{e +_b f \xrightarrow{\alpha|p} e'} \qquad \frac{\alpha \leq \bar{b} \qquad f \xrightarrow{\alpha|p} f'}{e +_b f \xrightarrow{\alpha|p} f'}$$

$$\frac{e \Rightarrow \alpha \qquad f \Rightarrow \alpha}{ef \Rightarrow \alpha} \qquad \frac{e \Rightarrow \alpha \qquad f \xrightarrow{\alpha|p} f'}{ef \xrightarrow{\alpha|p} f'} \qquad \frac{e \xrightarrow{\alpha|p} e'}{ef \xrightarrow{\alpha|p} e'f} \qquad \frac{\alpha \leq b \qquad e \xrightarrow{\alpha|p} e'}{e^{(b)} \xrightarrow{\alpha|p} e'e^{(b)}} \qquad \frac{\alpha \leq \bar{b}}{e^{(b)} \Rightarrow \alpha}$$

**Figure 3.2:** The transition structure of $(GExp, \delta)$. Here, $e, e', f, f' \in GExp$, $b \in BExp$, $\alpha \in At$, and $p \in \Sigma$. For a given $\alpha \in At$, if neither $e \Rightarrow \alpha$ nor $e \xrightarrow{\alpha|p} e'$ can be derived for any $p \in \Sigma$ and $e' \in GExp$, then $e \downarrow \alpha$. That is, transitions not explicitly defined are assumed to be failed termination.

**Definition 3.1.8.** A GKAT *automaton* [KT08; Smo+20] is a $G$-coalgebra[5] $(X, \delta_X)$, where for any set $X$ and any function $h\colon X \to Y$,

$$GX = (2 + \Sigma \times X)^{At} \qquad G(h)(\theta)(\alpha) = \begin{cases} \theta(\alpha) & \theta(\alpha) \in 2 \\ (p, h(x)) & \theta(\alpha) = (p, x) \in \Sigma \times X \end{cases}$$

for $\theta \in GX$ and $\alpha \in At$. Above, $At$ is the set of atomic tests (Definition 3.1.6), $2 = \{\bot, \top\}$, and $\Sigma$ is the set of primitive actions. We use $x \xrightarrow{\alpha|p}_{\delta_X} x'$ to denote $\delta(x)(\alpha) = (p, x')$, $x \Rightarrow_{\delta_X} \alpha$ to denote $\delta(x)(\alpha) = \top$ ($x$ *accepts* $\alpha$), and $x \downarrow_{\delta_X} \alpha$ to denote $\delta(x)(\alpha) = \bot$ ($x$ *rejects* $\alpha$). We also adopt the convention of writing $x \xrightarrow{b|p}_{\delta_X} x'$ for $b \in BExp$ to represent the multitude of transitions of the form $x \xrightarrow{\alpha|p} x'$ where $\alpha \leq b$. We drop the subscript $\delta$ when the automaton is clear from context.

Intuitively, $X$ represents the states of an abstract machine running a GKAT program with dynamics encoded in $\delta$. When the machine is in state $x$ and observes $\alpha \in At$, there are three possibilities: If $x \downarrow \alpha$, the machine rejects. If $x \Rightarrow \alpha$, it accepts. If $x \xrightarrow{\alpha|p} x'$, it performs the action $p$ and transitions to the state $x'$.

We can equip $GExp$ with a GKAT automaton structure that closely resembles the language interpretation of GKAT programs. It is called the *syntactic* GKAT *automaton* and written $(GExp, \delta)$, where $\delta$ is the transition map given by Brzozowski derivatives as specified in Figure 3.2.

The language semantics of GKAT programs coincides with the intuitive notion

[5] See Definition 2.2.6 for coalgebraic definitions.

of language acceptance in the syntactic GKAT automaton.

**Definition 3.1.9.** Let $(X, \delta_X)$ be a GKAT automaton and let $x \in X$. We write $\mathcal{L}(x, (X, \delta_X))$ for the guarded language *accepted* by $x$, defined

$$\mathcal{L}(x, (X, \delta_X)) = \left\{ \alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \;\middle|\; x \xrightarrow{\alpha_1 | p_1} x_1 \to \cdots \xrightarrow{\alpha_n | p_n} x_n \Rightarrow \alpha_{n+1} \right\}$$

Two states are called *language equivalent* if they accept the same guarded language. When the transition structure is clear, we typically write $\mathcal{L}(x)$ instead of $\mathcal{L}(x, (X, \delta_X))$.

**Lemma 3.1.10.** *For any $e \in GExp$, $\mathcal{L}(e) = \mathcal{L}(e, (GExp, \delta))$.*

*Proof.* By induction on $e$. In the base case, we have $\mathcal{L}(b) = \{\alpha \mid \alpha \leq b\} = \mathcal{L}(b, (GExp, \delta))$ for $b \in BExp$ and $\mathcal{L}(p) = \{\alpha_1 p \alpha_2 \mid \alpha_1, \alpha_2 \in At\} = \mathcal{L}(p, (GExp, \delta))$ for $p \in \Sigma$. For the inductive step, we have the following three cases.

- In the guarded union case, we have

  $$\mathcal{L}(e +_b f) = b \diamond \mathcal{L}(e) \cup \bar{b} \diamond \mathcal{L}(f) = b \diamond \mathcal{L}(e, (GExp, \delta)) \cup \bar{b} \diamond \mathcal{L}(f, (GExp, \delta))$$

  by the induction hypothesis. Now, given $w \in GS$, $\alpha p w \in \mathcal{L}(e +_b f, (GExp, \delta))$ if and only if either $\alpha \leq b$ and $\alpha p w \in \mathcal{L}(e, (GExp, \delta))$ or $\alpha \leq \bar{b}$ and $\alpha p w \in \mathcal{L}(f, (GExp, \delta))$. That is,

  $$b \diamond \mathcal{L}(e, (GExp, \delta)) \cup \bar{b} \diamond \mathcal{L}(f, (GExp, \delta)) = \mathcal{L}(e +_b f, (GExp, \delta))$$

- In the sequential composition case,

  $$\mathcal{L}(ef) = \mathcal{L}(e) \diamond \mathcal{L}(f) = \mathcal{L}(e, (GExp, \delta)) \diamond \mathcal{L}(f, (GExp, \delta))$$

  by the induction hypothesis. Let $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(e, (GExp, \delta)) \diamond \mathcal{L}(f, (GExp, \delta))$, and find an $i \leq n$ such that $\alpha_1 p_1 \cdots \alpha_i p_i \alpha_{i+1} \in \mathcal{L}(e, (GExp, \delta))$ and $\alpha_{i+1} p_{i+1} \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(f, (GExp, \delta))$. Then there are $e_1, \ldots, e_i, f_{i+1}, \ldots, f_n \in GExp$ such that

  $$e \xrightarrow{\alpha_1 | p_1} e_1 \to \cdots \xrightarrow{\alpha_i | p_i} e_i \Rightarrow \alpha_{i+1}$$
  $$f \xrightarrow{\alpha_{i+1} | p_{i+1}} f_{i+1} \to \cdots \xrightarrow{\alpha_n | p_n} f_n \Rightarrow \alpha_{n+1}$$

From the transition rules for $(GExp, \delta)$, this implies that

$$ef \xrightarrow{\alpha_1|p_1} e_1 f \to \cdots \xrightarrow{\alpha_i|p_i} e_i f \xrightarrow{\alpha_{i+1}|p_{i+1}} f_{i+1} \to \cdots \xrightarrow{\alpha_n|p_n} f_n \Rightarrow \alpha_{n+1}$$

Thus, $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(ef, (GExp, \delta))$. Conversely, if $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(ef, (GExp, \delta))$, then $ef \xrightarrow{\alpha_1|p_1} g_1 \to \cdots \xrightarrow{\alpha_n|p_n} g_n \Rightarrow \alpha_{n+1}$. By the transition rules, there is an $i \leq n$ such that for any $j \leq i$, $g_j = e_j f$ for some $e_j$, and for any $j > i$, $g_j = f_j$ for some $f_j$, such that

$$e \xrightarrow{\alpha_1|p_1} e_1 \to \cdots \xrightarrow{\alpha_i|p_i} e_i \Rightarrow \alpha_{i+1}$$
$$f \xrightarrow{\alpha_{i+1}|p_{i+1}} f_{i+1} \to \cdots \xrightarrow{\alpha_n|p_n} f_n \Rightarrow \alpha_{n+1}$$

This means $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(e, (GExp, \delta)) \diamond \mathcal{L}(f, (GExp, \delta))$. It follows that $\mathcal{L}(e, (GExp, \delta)) \diamond \mathcal{L}(f, (GExp, \delta)) = \mathcal{L}(ef, (GExp, \delta))$, as desired.

- In the guarded loop case,

$$\mathcal{L}(e^{(b)}) = \bigcup_{n \in \mathbb{N}} (b \diamond \mathcal{L}(e))^{\diamond n} \diamond \bar{b} = \bigcup_{n \in \mathbb{N}} (b \diamond \mathcal{L}(e, (GExp, \delta)))^{\diamond n} \diamond \bar{b}$$

by the induction hypothesis. If $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in (b \diamond \mathcal{L}(e, (GExp, \delta)))^{\diamond m} \diamond \bar{b}$, then we can find a partition of the indices $1 = i_0 \leq i_1 \leq \cdots \leq i_m = n$ such that $\alpha_{i_j} p_{i_j} \cdots \alpha_{i_{j+1}} p_{i_{j+1}} \alpha_{i_{(j+1)}+1} \in b \diamond \mathcal{L}(e, (GExp, \delta))$ for each $j < m$ and $\alpha_{n+1} p_{i_{m-1}} \cdots \alpha_n p_n \alpha_{n+1} \in b \diamond \mathcal{L}(e, (GExp, \delta)) \diamond \bar{b}$. That is, there exist $e_1, \ldots, e_n \in GExp$ such that $\alpha_1 \vee \alpha_{i_1} \vee \cdots \vee \alpha_{i_{m-1}} \leq b$, $\alpha_n \leq \bar{b}$,

$$e \xrightarrow{\alpha_1|p_1} e_1 \to \cdots \xrightarrow{\alpha_{i_1}|p_{i_1}} e_{i_1} \Rightarrow \alpha_{i_1+1}$$

and for $0 < j \leq m$,

$$e_{i_j - 1} \xrightarrow{\alpha_{i_j}|p_{i_j}} e_{i_j} \to \cdots \xrightarrow{\alpha_{i_{j+1}}|p_{i_{j+1}}} e_{i_{j+1}} \Rightarrow \alpha_{i_{(j+1)}+1}$$

By the transition rules for $(-)^{(b)}$,

$$e^{(b)} \xrightarrow{\alpha_1|p_1} e_1 e^{(b)} \to \cdots \xrightarrow{\alpha_{i_1}|p_{i_1}} e_{i_1} e^{(b)} \to \cdots \xrightarrow{\alpha_n|p_n} e_n e^{(b)} \Rightarrow \alpha_{n+1}$$

It follows that $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(e^{(b)}, (GExp, \delta))$. Conversely, suppose $e^{(b)} \xrightarrow{\alpha_1|p_1} g_1 \to \cdots \xrightarrow{\alpha_n|p_n} g_n \Rightarrow \alpha_{n+1}$. Using the transition rules for $(-)^{(b)}$ and sequential composition, there are expressions $e_1, \ldots, e_n$ such that $g_i = e_i e^{(b)}$ for each $i \leq n$, and furthermore for some $m \in \mathbb{N}$ there is a partition of the indices $1 = i_0 \leq i_1 \leq \cdots \leq i_m = n$ such that $\alpha_1 \vee \alpha_{i_1} \vee \cdots \vee \alpha_{i_{m-1}} \leq b$, $\alpha_n \leq \bar{b}$,

$$e \xrightarrow{\alpha_1|p_1} e_1 \to \cdots \xrightarrow{\alpha_{i_1}|p_{i_1}} e_{i_1} \Rightarrow \alpha_{i_1+1}$$

and for $0 < j \leq m$,

$$e_{i_j-1} \xrightarrow{\alpha_{i_j}|p_{i_j}} e_{i_j} \to \cdots \xrightarrow{\alpha_{i_{j+1}}|p_{i_{j+1}}} e_{i_{j+1}} \Rightarrow \alpha_{i_{(j+1)}+1}$$

It follows that $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in (b \diamond \mathcal{L}(e, (GExp, \delta)))^{\diamond m} \diamond \bar{b}$. □

*Remark* 3.1.11. An equivalent description of the syntactic GKAT automaton $(GExp, \delta)$ defines the function $\delta \colon GExp \to (2 + \Sigma \times GExp)^{At}$ as follows: For any $e, f \in GExp$, $b \in BExp$, $\alpha \in At$, and $p \in \Sigma$,

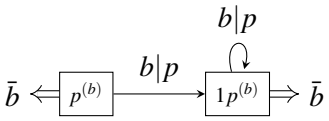
$$\delta(b)(\alpha) = \begin{cases} \top & \alpha \leq b \\ \bot & \alpha \leq \bar{b} \end{cases} \qquad \delta(p)(\alpha) = (p, 1) \qquad \delta(e +_b f) = \begin{cases} \delta(e)(\alpha) & \alpha \leq b \\ \delta(f)(\alpha) & \alpha \leq \bar{b} \end{cases}$$

$$\delta(ef)(\alpha) = \begin{cases} (p, e'f) & e \xrightarrow{\alpha|p} e' \\ \delta(f)(\alpha) & e \Rightarrow \alpha \\ \bot & \text{otherwise} \end{cases} \qquad \delta(e^{(b)})(\alpha) = \begin{cases} (p, e'e^{(b)}) & e \xrightarrow{\alpha|p} e' \text{ and } \alpha \leq b \\ \top & \alpha \leq \bar{b} \\ \bot & \text{otherwise} \end{cases}$$

*Remark* 3.1.12. Here we use the syntactic GKAT automaton as our operational model, defined analogously to Brzozowski's derivatives of regular expressions [Brz64]. In the original paper [Smo+20], the operational model of GKAT was given in terms of a Thompson-like construction. Models of GKAT programs generated by the syntactic GKAT automaton are bisimilar to automaton models given by the Thompson construction in [Smo+20]. See [Sch+21] for details.

*Example* 3.1.13. The operational interpretation of $p^{(b)}$ as a state of $(GExp, \delta)$ can be drawn as follows, where $x \xrightarrow{b|p} y$ denotes that $x \xrightarrow{\alpha|p} y$ for every $\alpha \leq b$. Rejecting transitions $- \downarrow \alpha$ are suppressed in the picture below, because they are uniquely

determined by the rest of the transitions.




(3.4)

**Lemma 3.1.14.** *For any $e \in GExp$, write $\langle e \rangle$ for the set* GKAT *expressions reachable from $GExp$ in $(GExp, \delta)$. Then $\langle e \rangle$ is a finite subcoalgebra of $(GExp, \delta)$.*

*Proof.* Let $\#(e)$ be the cardinality of the set of states in $\langle e \rangle$. We are going to show finiteness of $\#(e)$ by induction on $e$. The base cases are easily seen: $\#(b) = 1$ for $b \in BExp$ and $\#(p) = |\{p, 1\}| = 2$ for $p \in \Sigma$. For the inductive steps, we have

$$\begin{aligned}
\#(e +_b f) &\leq \Big| \{e +_b f\} \cup \bigcup \{\langle e_\alpha \rangle \mid e \xrightarrow{\alpha|p} e_\alpha \text{ and } \alpha \leq b\} \\
&\qquad\qquad \cup \bigcup \{\langle f_\alpha \rangle \mid f \xrightarrow{\alpha|p} f_\alpha \text{ and } \alpha \leq \bar{b}\} \Big| \\
&\leq |\{e +_b f\}| + \#(e) + \#(f) \\
\#(ef) &= \left| \{gf \mid g \in \langle e \rangle\} \cup \bigcup \left\{ \langle f_\alpha \rangle \mid e \xrightarrow{\alpha_1|p_1} \cdots \xrightarrow{\alpha_n|p_n} e_n \Rightarrow \alpha \text{ and } f \xrightarrow{\alpha|p} f_\alpha \right\} \right| \\
&\leq \#(e) + \#(f) \\
\#(e^{(b)}) &\leq \left| \{e^{(b)}\} \cup \left\{ ge^{(b)} \mid g \in \langle e \rangle \right\} \right| \\
&\leq |\{e^{(b)}\}| + \#(e)
\end{aligned}$$

□

*Remark* 3.1.15. In contrast to Lemma 3.1.14, Brzozowski derivatives of regular expressions do not produce finite automata [Brz64]. The canonical example of this is the regular expression $a^*$, whose Brzozowski derivatives are

$$a^* \xrightarrow{a} 0 + 1a^* \xrightarrow{a} 0 + (0 + 1a^*) \xrightarrow{a} 0 + (0 + (0 + 1a^*)) \xrightarrow{a} \cdots$$

The issue with finiteness here is, of course, the "0+" introduced at every step. The analogous GKAT program, on the other hand, has only one distinct derivative:

$$p^{(1)} \xrightarrow{1|p} 1p^{(1)} \xrightarrow{1|p} 1p^{(1)} \xrightarrow{1|p} \cdots$$

As this example illustrates, the GKAT transition rules do not introduce new syntax to expressions, in the sense that each of the derivatives of $e +_b f$, $ef$, and $e^{(b)}$ corresponds to a unique derivative of either $e$ or $f$. Every derivative of the program $e^{(b)}$ is of

the form $e'e^{(b)}$ for some derivative $e'$ of $e$, for example. This ensures that the GKAT automata generated by GKAT expressions are finite.

The automaton interpretation of GKAT programs suggests a slightly different notion of equivalence than the relational semantics does, especially if GKAT automata are seen as interactive machines (like the vending machines from (2.1)). In a setting where a user can interact with a GKAT program as it runs, states of the GKAT automaton are observationally equivalent if and only if reading the same sequence of atoms leads to the same sequence of actions, acceptance, or rejection in each state. This happens when one state mimics the moves of the other, performing the same actions in response to the same stimuli.

For example, consider the GKAT automaton in (3.4). The behaviour of $p^{(b)}$ can be replicated by the behaviour of $1 \cdot p^{(b)}$, in that both either consume an $\alpha \leq \bar{b}$ and terminate or consume $\alpha \leq b$ and emit $p$ before transitioning to $1 \cdot p^{(b)}$. Again, the appropriate notion of equivalence is obtained by instantiating the coalgebraic notion of bisimulation in Definition 2.2.8 to GKAT automata.

**Lemma 3.1.16.** *Let $R \subseteq X \times Y$ be a relation between the state spaces of* GKAT *automata $(X, \delta_X)$ and $(Y, \delta_Y)$. Then $R$ is a bisimulation if and only if for any $(x, y) \in R$ and $\alpha \in At$,*

1. *$x \downarrow \alpha$ if and only if $y \downarrow \alpha$,*
2. *$x \Rightarrow \alpha$ if and only if $y \Rightarrow \alpha$, and*
3. *if $x \xrightarrow{\alpha|p} x'$ and $y \xrightarrow{\alpha|q} y'$ for some $x'$ and $y'$, then $p = q$ and $(x', y') \in R$.*

Moreover, the functor $G$ weakly preserves pullbacks (Definition 2.2.11) [Gum98], so bisimilarity is an equivalence relation and $G$-coalgebra homomorphisms coincide with functional bisimulations (see Lemma 2.2.7).

### The final GKAT automaton

Interestingly, the bisimulation equivalence class of a GKAT program can be concretely identified with a certain kind of tree that tracks acceptance, rejection, and transitions to other states (which are represented as subtrees). Roughly, one can obtain the tree corresponding to a state of a GKAT automaton by unfolding the transition graph from that state. The trees obtained this way are states of a GKAT automaton $(Z, \zeta)$, which turns out to be *final* in the sense described in Definition 2.5.1.

Write $At^+$ for the set of all non-empty words consisting of atoms and $\mathrm{dom}(t)$ for the domain of a partial function $t : X \rightharpoonup Y$.

**Definition 3.1.17.** A *(*GKAT*) behaviour tree* is a partial function $t : At^+ \rightharpoonup 2 + \Sigma$ with $At \subseteq \mathrm{dom}(t)$, such that the following hold for all $\alpha \in At$ and $v \in At^+$:

$$\frac{w \in \mathrm{dom}(t) \quad t(w) \in \Sigma}{w\alpha \in \mathrm{dom}(t)} \qquad \frac{w \in \mathrm{dom}(t) \quad t(w) \in 2}{wv \notin \mathrm{dom}(t)}$$

We write $Z$ for the set of all behaviour trees. Let $\partial_\alpha t = \lambda w.t(\alpha w)$ for any $\alpha \in At$. The transition structure $\zeta$ of $(Z, \zeta)$ is defined by

$$\zeta(t)(\alpha) = \begin{cases} t(\alpha) & t(\alpha) \in 2 \\ (t(\alpha), \partial_\alpha t) & t(\alpha) \in \Sigma \end{cases}$$

We think of $t \in Z$ as a (graphical) tree where the root has leaves for atoms $\alpha \in At$ with $t(\alpha) = \top$, and a subtree for every $\alpha \in At$ with $t(\alpha) \in \Sigma$.

*Remark* 3.1.18. Trees correspond to *deterministic infinitary languages* [Smo+20; KT08]. More precisely, every tree can be identified with a language $L \subseteq GS^\infty = (At \cdot \Sigma)^* \cdot At \cup (At \cdot \Sigma)^\omega$ satisfying the following two conditions: For $w \in (At \cdot \Sigma)^*$, $\alpha \in At$, $p, q \in \Sigma$, and $\sigma, \sigma' \in GS^\infty$,

1. if $w\alpha p\sigma, w\alpha q\sigma' \in L$, then $p = q$, and
2. if $w\alpha \in L$, then $w\alpha p\sigma \notin L$ for any $p\sigma$.

See [Smo+20, Appendix C] for details. We forgo a description in terms of guarded languages in favour of trees, because trees have the determinism constraint built-in.

**Theorem 3.1.19.** *$(Z, \zeta)$ is a final* GKAT *automaton.*

*Proof.* Let $(X, \delta_X)$ be a GKAT automaton. We define an unrolling map $!_{\delta_X} : X \to Z$ as follows: For any $\alpha_1 \cdots \alpha_n \in At^+$, let

$$!_{\delta_X}(x)(\alpha_1 \cdots \alpha_n) = \begin{cases} p_n & x \xrightarrow{\alpha_1|p_1} x_1 \cdots \xrightarrow{\alpha_n|p_n} x_n \\ \top & x \xrightarrow{\alpha_1|p_1} x_1 \cdots \xrightarrow{\alpha_{n-1}|p_{n-1}} x_{n-1} \Rightarrow \alpha_n \\ \bot & x \xrightarrow{\alpha_1|p_1} x_1 \cdots \xrightarrow{\alpha_{n-1}|p_{n-1}} x_{n-1} \downarrow \alpha_n \\ \text{undefined} & \text{otherwise} \end{cases}$$

To see that $!_{\delta_X}$ is a coalgaebra homomorphism, take $\alpha \in At$. There are three cases to consider: First, if $x \downarrow \alpha$, then

$$\zeta(!_{\delta_X}(x))(\alpha) = !_{\delta_X}(x)(\alpha) = \bot = G(!_{\delta_X})(\delta_X(x))(\alpha)$$

If $x \Rightarrow \alpha$, then similarly

$$\zeta(!_{\delta_X}(x))(\alpha) = !_{\delta_X}(x)(\alpha) = \top = G(!_{\delta_X})(\delta_X(x))(\alpha)$$

Lastly, if $x \xrightarrow{\alpha|p} y$, then

$$\begin{aligned}\zeta(!_{\delta_X}(x))(\alpha) &= (!_{\delta_X}(x)(\alpha), \lambda w.!_{\delta_X}(x)(\alpha w)) \\ &= (!_{\delta_X}(x)(\alpha), !_{\delta_X}(y)) \\ &= (p, !_{\delta_X}(y)) \\ &= G(!_{\delta_X})(\delta_X(x))(\alpha)\end{aligned}$$

To see that $!_{\delta_X}$ is the unique coalgebra homomorphism into $(Z,\zeta)$, let $h\colon (X,\delta_X) \to (Z,\zeta)$ be any other homomorphism. We show that $!_{\delta_X}(x)(\alpha_1 \cdots \alpha_n) = h(x)(\alpha_1 \cdots \alpha_n)$ for all $x \in X$ by induction on $n$. For the base case, let $x \in X$ and $\alpha_1 \in At$. If $x \downarrow \alpha_1$, then

$$!_{\delta_X}(x)(\alpha_1) = \bot = G(h)(\delta_X(x))(\alpha_1) = \zeta(h(x))(\alpha_1) = h(x)(\alpha_1)$$

Similarly, $x \Rightarrow \alpha$ implies $!_{\delta_X}(x)(\alpha_1) = \top$ and $h(x)(\alpha_1) = \top$. If $x \xrightarrow{\alpha_1|p_1} y$, then

$$!_{\delta_X}(x)(\alpha_1) = p = \pi_1 \circ G(h)(\delta_X(x))(\alpha_1) = \pi_1 \circ \zeta(h(x))(\alpha_1) = h(x)(\alpha_1)$$

where $\pi_1(p,y) = p$. This establishes the base case. For the inductive step, let $x = x_1 \xrightarrow{\alpha_1|p_1} \cdots \xrightarrow{\alpha_n|p_n} x_{n+1}$ as assume that $!_{\delta_X}(x_i) = h(x_i)$ for $i \leq n$. By the induction hypothesis, we know that $\alpha_1 \ldots \alpha_{n+1}$ is in both $\mathrm{dom}(!_{\delta_X}(x))$ and $\mathrm{dom}(h(x))$ (see the first rule in Definition 3.1.17). Furthermore,

$$!_{\delta_X}(x)(\alpha_1 \cdots \alpha_n\alpha_{n+1}) = !_{\delta_X}(x_n)(\alpha_n\alpha_{n+1}) = h(x_n)(\alpha_n\alpha_{n+1}) = h(x)(\alpha_1 \cdots \alpha_{n+1})$$

The second equality follows from the induction hypothesis applied to $x_n$. The first and

last equalities hold because $!_{\delta_X}$ and $h$ are coalgebra homomorphisms respectively. □

It follows from Lemma 2.5.4 that for states $x, y$ of a GKAT automaton $(X, \delta_X)$, $x \leftrightarrows y$ ($x$ and $y$ are bisimilar) if and only if $!_{\delta_X}(x) = !_{\delta_X}(y)$. It is furthermore true that if $!_{\delta_X}(x) = !_{\delta_X}(y)$, then $x$ and $y$ are language equivalent. If $\delta_X$ is clear from context, we usually omit it and write $!(x)$ instead of $!_{\delta_X}(x)$.

**Lemma 3.1.20.** *If a two states are bisimilar, then they are language equivalent. That is, if $(X, \delta_X)$ is a* GKAT *automaton and $x, y \in X$, then $!(x) = !(y)$ implies $\mathcal{L}(x) = \mathcal{L}(y)$.*

*Proof.* Let $R$ be a bisimulation between GKAT-automata $(X, \delta_X), (Y, \delta_Y)$. To show that $\mathcal{L}(x) = \mathcal{L}(y)$ for any $(x, y) \in R$, we use Lemma 3.1.16 and show that $w \in \mathcal{L}(x)$ iff $w \in \mathcal{L}(y)$ by induction on the length of $w$.

Given $(x, y) \in R$ and $\alpha \in At$, $x \Rightarrow \alpha$ if and only if $y \Rightarrow \alpha$ by the second property in Lemma 3.1.16. Hence, $\alpha \in \mathcal{L}(x)$ if and only if $\alpha \in \mathcal{L}(y)$.

Now let $n > 1$ and assume that for any $(x, y) \in R$ and $w = \alpha_1 p_1 \cdots \alpha_{n-1} p_{n-1} \alpha_n$, $w \in \mathcal{L}(x)$ if and only if $w \in \mathcal{L}(y)$. If $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(x)$, then there is an $x' \in X$ such that $x \xrightarrow{\alpha_1 | p_1} x'$ and $\alpha_2 p_2 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(x')$. By the third property in Lemma 3.1.16, $y \xrightarrow{\alpha_1 | p_1} y'$ in $(Y, \delta_Y)$ for some $y'$ such that $(x', y') \in R$. By the induction hypothesis, $\alpha_2 p_2 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(y')$, and therefore $\alpha_1 p_1 \cdots \alpha_n p_n \alpha_{n+1} \in \mathcal{L}(y)$. Thus, $\mathcal{L}(x) \subseteq \mathcal{L}(y)$. By symmetry, $\mathcal{L}(x) = \mathcal{L}(y)$. □

On the other hand, it is not the case that two states of a GKAT automaton are bisimilar whenever they are language equivalent. For example, $p0$ and $0$ are language equivalent, since neither accepts any guarded traces. However, $p0$ makes an outgoing transition $p0 \xrightarrow{1|p} 0$, whereas $0$ makes no outgoing transitions. The issue is that $p0$ has a *dead branch* that is not equivalent to $0$. Language equivalent but nonbisimilar expressions can be transformed into bisimilar expressions by systematically replacing dead branches by $0$. We discuss this point in more detail in Section 3.1.5.

### 3.1.4 Sound Program transformations in GKAT

GKAT programs are a variation on regular expressions, which are well-studied in computer science and intuitive to reason about. In [Smo+20], a set of axioms $e = f$ is proposed such that $\mathcal{L}(e) = \mathcal{L}(f)$, and it is shown that these can be used to prove a number of useful facts about programs. The axioms presented in [Smo+20] (with

$$\begin{array}{llll} b \vee 0 = b & b \vee \bar{b} = 1 & b \vee c = c \vee b & b \vee (c \wedge d) = (x \vee c) \wedge (b \vee d) \\ b \wedge 1 = b & b \wedge \bar{b} = 0 & b \wedge c = c \wedge b & b \wedge (c \vee d) = (b \wedge c) \vee (b \wedge d) \end{array}$$

**Figure 3.3:** The axioms of Boolean algebra (BA) [Hun04]. We write $b =_{\mathsf{BA}} c$ if $\mathsf{BA} \vdash b = c$.

$$\begin{array}{lll} \text{(U1)} & e +_b e = e & \\ \text{(U2)} & e_1 +_b e_2 = e_2 +_{\bar{b}} e_1 & \\ \text{(U3)} & (e_1 +_b e_2) +_c e_3 = e_1 +_{b \wedge c} (e_2 +_c e_3) & \\ \text{(U4)} & e_1 +_b e_2 = be_1 +_b e_2 & \\ \text{(U5)} & (e_1 +_b e_2)e_3 = e_1 e_3 +_b e_2 e_3 & \\ \text{(W1)} & e_1{}^{(b)} e_2 = e_1 (e_1{}^{(b)} e_2) +_b e_2 & \\ \text{(W2)} & (e +_c 1)^{(b)} = (ce)^{(b)} & \end{array} \qquad \begin{array}{ll} \text{(S1)} & e_1(e_2 e_3) = (e_1 e_2)e_3 \\ \text{(S2)} & 0e = 0 \\ \text{(S3)} & e0 = 0 \\ \text{(S4)} & 1e = e \\ \text{(S5)} & e1 = e \\ \text{(S6)} & bc = b \wedge c \\ \text{(W3)} & \dfrac{g = e_1 g +_b e_2 \quad E(e_1) = 0}{g = e_1{}^{(b)} e_2} \end{array}$$

**Figure 3.4:** The axiom system for language equivalence of GKAT expressions. Above, $e, e_1, e_2, e_1, e_2 \in \mathit{GExp}$ and $b, c \in \mathit{BExp}$. As a theory, GKAT consists of the axioms of Boolean algebra (see Figure 3.3), (U1)-(U5), (S1)-(S6), and (W1)-(W3). The theory $\mathsf{GKAT}_0$ consists of the axioms BA of Boolean algebra, (U1)-(U5), (S1), (S2), (S4)- (S6), and (W1)-(W3).

the added axiom (S6) from [Sch+21], which is needed because $bc$ and $b \wedge c$ are syntactically different tests) are given in Figure 3.4.

For instance, the axiom (U5) says that common code at the tail end of branches can be factored out, and (U2) says that the code in branches of a conditional can be swapped as long as we negate the test. Returning to our running example, we can apply (U5) to (3.2) three times (once for each guarded choice).

$$(n := 1) \left( \left( \begin{array}{c} (\mathit{fizzbuzz} +_{5|n} \mathit{fizz}) +_{3|n} \\ (\mathit{buzz} +_{5|n} n) \end{array} \right) (n{+}{+}) \right)^{(n \leq 100)} (\mathit{done!}) \tag{3.5}$$

Then we can apply $(e +_b f) +_c (g +_b h) = e +_{b \wedge c} (f +_c (g +_b h))$, which is provable from the axioms of GKAT (see Proposition 3.2.15), to transform (3.5) into (3.3).

**Definition 3.1.21.** We define the theory GKAT to be the union of the axioms in Figure 3.4 with the axioms of Boolean algebra (applied to $\mathit{BExp}$). The theory $\mathsf{GKAT}_0$, also called *bisimulation* GKAT, is GKAT without (S3).

(G0) $e +_1 f = e$

(G1) $e +_b e = e$

(G2) $e_1 +_b e_2 = e_2 +_{\bar{b}} e_1$

(G3) $e_1 +_b (e_2 +_c e_3) = (e_1 +_b e_2) +_{b \vee c} e_3$

(G4) $e1 = e$

(G5) $1e = e$

(G6) $0e = 0$

(G7) $e_1(e_2e_3) = (e_1e_2)e_3$

(G8) $(e_1 +_b e_2)e_3 = e_1e_3 +_b e_2e_3$

(FP1) $e_1 {}^{(b)}e_2 = e_1(e_1 {}^{(b)}e_2) +_b e_2$

(FP2) $(e +_c 1)^{(b)} = (ce)^{(b)}$

(RSP*) $\dfrac{g = e_1 g +_b e_2 \quad E(e_1) = 0}{g = e_1 {}^{(b)}e_2}$

(R0) $e0 = 0$

(DM) $b = 1 +_b 0$

**Figure 3.5:** An equivalent axiom system for language equivalence of GKAT expressions. Here, $e, e_1, e_2, e_1, e_2 \in GExp$ and $b, c \in BExp$. As a theory, GKAT is equivalent to the axioms of Boolean algebra (see Figure 3.3), (G0)-(G8), (FP1), (FP2), (R0), (DM), and (RSP*). The theory $\mathsf{GKAT}_0$ is equivalent to the axioms BA of Boolean algebra, (G0)-(G8), (FP1), (FP2), (DM), and (RSP*).

### An alternative axiomatization of GKAT

In later sections, it will be convenient[6] for us to use a different but equivalent set of axioms for GKAT and $\mathsf{GKAT}_0$.

**Proposition 3.1.22.** *Let $e, f \in GExp$. Then $\mathsf{GKAT} \vdash e = f$ if and only if $e = f$ is provable from the axioms in Figure 3.5, and $\mathsf{GKAT}_0 \vdash e = f$ if and only if $e = f$ is provable from the axioms in Figure 3.5 without (R0).*

*Proof.* The axioms (G1) and (G2) are (U1) and (U2) respectively, and (G4)-(G8) are (S5), (S4), (S2), (S1), and (U5) respectively. The axioms (FP1), (FP2), and (RSP*) are (W1)-(W3) respectively. Since (S3) is (R0), it suffices to show that $\mathsf{GKAT}_0$ is equivalent to Figure 3.5 without (R0). This leaves us with the following proof obligation: We need to show that (U3), (U4), and (S6) are derivable from Figure 3.5 without (S3), and that (G0), (G3), and (DM) are derivable in $\mathsf{GKAT}_0$.

To derive (U3),

$$(e +_b f) +_c g \overset{(G2)}{=} g +_{\bar{c}} (f +_{\bar{c}} e) \overset{(G3)}{=} (g +_{\bar{c}} f) +_{\bar{b} \vee \bar{c}} e \overset{(G2)}{=} e +_{\overline{\bar{b} \vee \bar{c}}} (f +_{\bar{c}} g) \overset{(BA)}{=} e +_{b \wedge c} (f +_c g)$$

(G3) can similarly be derived from (U2), (U3), and (BA). The axiom (U4) $e +_b f =$

[6] Figure 3.5 conveniently restricts to a complete axiomatization of the skip-free fragment, as we will see in Section 3.2.

$be +_b f$ is derivable from Figure 3.5 as follows:

$$be +_b f \overset{(DM)}{=} (1 +_b 0)e +_b f \overset{(G8)}{=} (1e +_b 0e) +_b f \overset{(G5,6)}{=} (e +_b 0) +_b f \overset{(*)}{=} e +_b f$$

The identity (*) can be derived from Figure 3.5 and (U2) (which we have just derived from (G2) and (G3)) as follows:

$$\begin{aligned} e +_b f &\overset{(G0)}{=} (e +_1 0) +_b f \overset{(U2)}{=} e +_{b\wedge 1} (0 +_b f) \overset{(BA)}{=} e +_b (0 +_b f) \\ &\overset{(G3)}{=} (e +_b 0) +_{b\vee b} f \overset{(BA)}{=} (e +_b 0) +_b f \end{aligned}$$

The identity (S6) is derived from Figure 3.5 as follows:

$$\begin{aligned} bc &\overset{(DM)}{=} (1 +_b 0)(1 +_c 0) \overset{(G8,5,6)}{=} (1 +_c 0) +_b 0 \\ &\overset{(U2)}{=} 1 +_{b\wedge c} (0 +_b 0) \overset{(G1)}{=} 1 +_{b\wedge c} 0 \overset{(DM)}{=} b \wedge c \end{aligned}$$

It remains to derive (G0) and (DM) in $\mathsf{GKAT}_0$. Both derivations are short calculations.

$$\begin{aligned} e +_1 f &\overset{(U2)}{=} f +_0 e \overset{(U4)}{=} 0f +_0 e \overset{(S2)}{=} 0 +_0 e \overset{(S2)}{=} 0e +_0 e \overset{(U4)}{=} e +_0 e \overset{(U1)}{=} e \\ b &\overset{(U1)}{=} b +_b b \overset{(S5)}{=} b1 +_b b \overset{(U2,U4)}{=} b1 +_b \bar{b}b \overset{(BA)}{=} b1 +_b 0 \overset{(U4)}{=} 1 +_b 0 \end{aligned}$$ □

As a result of Proposition 3.1.22, we are going to derive equivalences in GKAT and $\mathsf{GKAT}_0$ using Figure 3.5 instead of Figure 3.4 from now on.

**A note about side conditions**

Both Figure 3.4 and Figure 3.5 include an axiom that, much like Salomaa's axiom (RSP*) in Figure 2.1, requires a side-condition in terms of a function $E$. The function $E : \mathit{GExp} \to \mathit{BExp}$ is defined by

$$E(b) = b \qquad E(p) = 0 \qquad E(e_1 +_b e_2) = (b \wedge E(e_1)) \vee (\bar{b} \wedge E(e_2))$$
$$E(e_1 \cdot e_2) = E(e_1) \wedge E(e_2) \qquad E(e^{(b)}) = \bar{b}$$

The side condition $E(0) = 0$ in (RSP*) in Figure 2.1 is called the *empty word property* by Salomaa, as it determines whether an expression represents an accepting state. Similarly, $E(e) = 0$ follows from the axioms of Boolean algebra if and only if $\neg(e \Rightarrow \alpha)$

for any $\alpha \in At$. The side condition $E(e) =_{\mathsf{BA}} 0$ is necessary for the soundness of (RSP$^*$): Let $b \neq 0$ and observe that $\mathsf{GKAT}_0 \vdash 1 = 1 \cdot 1 +_b 1$. The conclusion of (RSP$^*$) would then say that $\mathsf{GKAT}_0 \vdash 1 = 1^{(b)}$, but $\mathcal{L}(1) \neq \bar{b} = \mathcal{L}(1^{(b)})$.

**Theorem 3.1.23** (Soundness)**.** *Let $e, f \in GExp$.*

1. *([Sch+21]) If $\mathsf{GKAT}_0 \vdash e = f$, then $!(e) = !(f)$.*
2. *([Smo+20]) If $\mathsf{GKAT} \vdash e = f$, then $\mathcal{L}(e) = \mathcal{L}(f)$.*

*Proof.* By an induction on the proof of $\mathsf{GKAT}_0 \vdash e = f$, one can show that the congruence relation induced by bisimulation GKAT satisfies the conditions of being a bisimulation on $(GExp, \delta)$ from Lemma 3.1.16.[7] This establishes *1.* To see *2.*, observe that $\mathcal{L}(e \cdot 0) = \mathcal{L}(e) \diamond \mathcal{L}(0) = \mathcal{L}(e) \diamond \emptyset = \emptyset = \mathcal{L}(0)$, so that $e \cdot 0 = 0$ is sound for language equivalence. □

### 3.1.5 Completeness of Bisimulation GKAT with UA

Being able to transform one GKAT program into another using the axioms of GKAT is useful, but the question arises: Do the axioms capture *all* equivalences that hold? In [Smo+20], a partial answer to this question is provided: If we extend GKAT with an axiom called the *uniqueness axiom* (UA), then the resulting set of axioms is sound and complete with respect to the language semantics. The problem with this is that UA is not really a single axiom, but rather an *axiom scheme*, which makes both its presentation and application somewhat unwieldy.

**Definition 3.1.24.** Given $e_{ij}, f_i \in GExp$ such that $E(e_{ij}) =_{\mathsf{BA}} 0$ and $b_{ij} \in BExp$ for $i, j \in \{1, \ldots, n\}$, the *uniqueness axiom for* $\{(e_{ij}, f_{ij}, b_{ij})\}_{i,j \leq n}$ is the inference rule

$$\frac{\{g_i = e_{i1} g_1 +_{b_{i1}} e_{i2} g_2 +_{b_{i2}} \cdots +_{b_{in}} f_{i1}\}_{i \leq n} \quad \{h_i = e_{i1} h_1 +_{b_{i1}} e_{i2} h_2 +_{b_{i2}} \cdots +_{b_{in}} f_{i2}\}_{i \leq n}}{g_1 = h_1}$$

We use UA, or *the uniqueness axiom*, to refer to the set consisting of all uniqueness axioms for any $\{(e_{ij}, f_{ij}, b_{ij})\}_{i,j \leq n}$ with $E(e_{ij}) =_{\mathsf{BA}} 0$ for all $i, j \leq n$. Given $e, f \in GExp$,

$$e \equiv f \text{ denotes that } \mathsf{GKAT} + \mathsf{UA} \vdash e = f$$

$$e \equiv_0 f \text{ denotes that } \mathsf{GKAT}_0 + \mathsf{UA} \vdash e = f$$

[7] Alternatively, the soundness of GKAT with respect to behavioural equivalence is a special case of the soundness theorem that appears directly after Theorem 4.5.5 in Chapter 4.

A system of equations of the form

$$\{x_i = e_{i1}x_1 +_{b_{i1}} e_{i2}x_2 +_{b_{i2}} \cdots +_{b_{in}} f_i\}_{i\leq n} \tag{3.6}$$

is called a *left-affine system* if $b_{ij} \wedge b_{ik} =_{\mathsf{BA}} 0$ for any $i \leq n$ and $j < k \leq n$. If we furthermore have $E(e_{ij}) =_{\mathsf{BA}} 0$ for all $i, j \leq n$, then the left-affine system above is called *Salomaa*. Given an equivalence relation $\approx$ on *GExp*, a $\approx$*-solution* to (3.6) is a function $\varphi\colon \{x_1, \ldots, x_n\} \to GExp$ such that

$$\varphi(x_i) \approx e_{i1}\varphi(x_1) +_{b_{i1}} e_{i2}\varphi(x_2) +_{b_{i2}} \cdots +_{b_{in}} f_i$$

for each $i \leq n$. Two solutions $\varphi_1, \varphi_2$ are $\approx$*-equivalent*, written $\varphi_1 \approx \varphi_2$, if $\varphi_1(x_i) \approx \varphi_2(x_i)$ for each $i \leq n$. An alternative statement of the uniqueness axiom is that every left-affine system of equations has at most one $\approx$-solution up to $\approx$-equivalence.

**Theorem 3.1.25.** *The uniqueness axiom is sound for both the behaviour and language semantics of* GKAT. *That is, if $e \equiv_0 f$, then $!(e) = !(f)$, and if $e \equiv f$, then $\mathcal{L}(e) = \mathcal{L}(f)$.*

*Proof.* To prove the soundness of $\equiv_0$ with respect to the behaviour tree interpretation of GKAT expressions, we extend the proof of soundness of $\mathsf{GKAT}_0$ (Theorem 3.1.23), a proof by induction on the derivation of $\mathsf{GKAT}_0 \vdash e = f$, with an additional inductive step for the UA. This proves that $\equiv_0$ is a bisimulation equivalence, which implies the first soundness result Lemma 2.5.4 and Theorem 3.1.19.

We are left with showing that for any two $\equiv_0$-solutions $\varphi_1, \varphi_2$ to a Salomaa system $S$, $(\varphi_1(x_i), \varphi_2(x_i))$ satisfies the three properties of a pair in a bisimulation in Lemma 3.1.16. To this end, suppose that $\varphi_1, \varphi_2$ are two $\equiv_0$-solutions to the Salomaa system of equations

$$S = \{x_i = e_{i1}x_1 +_{b_{i1}} \cdots +_{b_{in}} f_i\}_{i\leq n}$$

and let $\varphi_1(x_i) = g_i$ and $\varphi_2(x_i) = h_i$ for $i \leq n$. The induction hypothesis says that both $(g_i, e_{i1}g_1 +_{b_{i1}} \cdots +_{b_{in}} f_i)$ and $(h_i, e_{i1}h_1 +_{b_{i1}} \cdots +_{b_{in}} f_i)$ satisfy the three properties of a pair in a bisimulation in Lemma 3.1.16. We need to show that for any $\alpha \in At$, $i \leq n$, and $p \in \Sigma$, (i) $g_i \downarrow \alpha$ if and only if $h_i \downarrow \alpha$, (ii) $g_i \Rightarrow \alpha$ if and only if $h_i \Rightarrow \alpha$, and (iii) if $g_i \xrightarrow{\alpha|p} g'$, then $h_i \xrightarrow{\alpha|p} h'$ for some $h'$ such that $g' \equiv_0 h'$.

(i) Suppose $g_i \downarrow \alpha$. By the induction hypothesis, $e_{i1}g_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \downarrow \alpha$. Since $S$ is

Salomaa, either $\alpha \leq b_{ij}$ and $e_{ij} \downarrow \alpha$ for some $j$, or $\alpha \leq \bar{b}_{in}$ and $f_i \downarrow \alpha$. In either case, $e_{i1}h_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \downarrow \alpha$ and by the induction hypothesis $h_i \downarrow \alpha$ as well. The converse is similar.

(ii) Suppose $g_i \Rightarrow \alpha$. By the induction hypothesis, $e_{i1}g_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \Rightarrow \alpha$. Since $S$ is Salomaa, this means that $\alpha \leq \bar{b}_{in}$ and $f_i \Rightarrow \alpha$. This implies that $e_{i1}h_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \Rightarrow \alpha$, so by the induction hypothesis, $h_i \Rightarrow \alpha$. The converse is similar.

(iii) Now suppose $g_i \xrightarrow{\alpha|p} g'$. By the induction hypothesis, $e_{i1}g_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \xrightarrow{\alpha|p} g''$ for some $g'' \equiv_0 g'$. Since $S$ is Salomaa, there are two possibilities: Either (a) $\alpha \leq b_{ij}$ and $e_{ij} \xrightarrow{\alpha|p} e'$ and $g'' = e'g_j$ for some $j$, or (b) $\alpha \leq \bar{b}_{in}$ and $f_i \xrightarrow{\alpha|p} g''$.

(a) In this case, we also have $e_{ij}h_j \xrightarrow{\alpha|p} e'h_j$. By the induction hypothesis, $h_i \xrightarrow{\alpha|p} h'$ for some $h' \equiv_0 e'h_j$. Since $g_j \equiv_0 h_j$, $g' \equiv_0 e'g_j \equiv_0 e'h_j \equiv_0 h'$ by congruence. By transitivity, $g' \equiv_0 h'$.

(b) In this case, $e_{i1}h_1 +_{b_{i1}} \cdots +_{b_{in}} f_i \xrightarrow{\alpha|p} g''$ as well, so by the induction hypothesis there is an $h' \equiv_0 g''$ such that $h_i \xrightarrow{\alpha|p} h'$. This concludes this case, as $g' \equiv_0 g'' \equiv_0 h'$.

This concludes the proof that $\equiv_0$ is a bisimulation equivalence on $(GExp, \delta)$. Soundness then follows from Lemma 2.5.4.

Soundness with respect to language equivalence is [Smo+20, Theorem 6.2]. □

As we will see next, when added to bisimulation GKAT, the uniqueness axiom enables a proof of completeness that resembles Salomaa's proof of completeness for his first axiomatization of regular algebra [Sal66]. While our completeness theorem characterizes bisimilarity of GKAT expressions (and not language equivalence), Salomaa's completeness proof is for language equivalence of regular expressions. This apparent contrast is less surprising than it first appears, as language equivalence of DFAs happens to coincide with bisimilarity of DFAs.

### Solutions and homomorphisms

One can associate every finite GKAT automaton with a system of equations as follows: Given any finite GKAT automaton $(X, \delta_X)$, we obtain the left-affine system

$$\mathcal{S}(X, \delta_X) = \{x_i = p_{i1}x_1 +_{b_{i1}} p_{i2}x_2 +_{b_{i2}} \cdots +_{b_{in}} c_i\}_{i \leq n}$$

where $X = \{x_1, \ldots, x_n\}$, and for any $i, j \leq n$, $x_i \xrightarrow{b_{ij}|p_{ij}} x_j$ and $x_i \Rightarrow c_i$. Observe that the left-affine system associated with a GKAT automaton is always Salomaa.

Recall from the proof of soundness (Theorem 3.1.25) that $\equiv_0$ is a bisimulation equivalence. By Lemma 2.2.12 item 3, it is the kernel of a coalgebra homomorphism $[-]_{\equiv_0} : (GExp, \delta) \to (GExp/\equiv_0, [\delta]_{\equiv_0})$. The next result (Lemma 3.1.26) exposes the close relationship between coalgebra homomorphisms of the form $[-]_{\equiv_0} \circ \varphi$ and provable equivalence. It is a special case of Lemma 4.5.10.

**Lemma 3.1.26.** *Let $(X, \delta_X)$ be a finite* GKAT *automaton with associated system of equations $\mathcal{S}$. A map $\varphi : X \to GExp$ is a $\equiv_0$-solution to $\mathcal{S}$ if and only if $[-]_{\equiv_0} \circ \varphi$ is a G-coalgebra homomorphism $(X, \delta_X) \to (GExp/\equiv_0, [\delta]_{\equiv_0})$.*

In particular, the inclusion homomorphism $\langle e \rangle \hookrightarrow (GExp, \delta)$ is a solution to the system associated with $e$, for every $e \in GExp$.

Lemma 3.1.26 is the key lemma needed to prove completeness of bisimulation GKAT with the uniqueness axiom. Given any two homomorphisms $\varphi_1, \varphi_2 : (X, \delta_X) \to (GExp, \delta)$ from a finite GKAT automaton $(X, \delta_X)$, we know that $[-]_{\equiv_0} \circ \varphi_1, [-]_{\equiv_0} \circ \varphi_2$ are also homomorphisms. By Lemma 3.1.26, $\varphi_1, \varphi_2$ are $\equiv_0$-solutions to the left-affine system associated with $(X, \delta_X)$, so by the uniqueness axiom, $\varphi_1(x) \equiv_0 \varphi_2(x)$ for any $x \in X$. This is equivalent to saying that $[-]_{\equiv_0} \circ \varphi_1 = [-]_{\equiv_0} \circ \varphi_2$.

**Theorem 3.1.27** (Completeness I)**.** *For any $e, f \in GExp$, if $!(e) = !(f)$, then $e \equiv_0 f$.*

*Proof.* Suppose $!(e) = !(f)$, and recall that this is equivalent to $e \leftrightarroweq f$. That is, there is a bisimulation $(R, \delta_R)$ between $\langle e \rangle$ and $\langle f \rangle$ with $(e, f) \in R$. From the discussion above, we know that the GKAT automaton homomorphisms $[-]_{\equiv_0} \circ \pi_1, [-]_{\equiv_0} \circ \pi_2 : (R, \delta_R) \to (GExp/\equiv_0, [\delta]_{\equiv_0})$ are equal. In particular, $[e]_{\equiv_0} = [f]_{\equiv_0}$, or equivalently, $e \equiv_0 f$. □

### 3.1.6 Completeness of GKAT with UA

Theorem 3.1.27 establishes the completeness of bisimulation GKAT with UA as an axiomatization of bisimilarity of GKAT programs. In this section, we reduce the completeness of GKAT with UA, the axiomatization of language equivalence of GKAT programs, to the completeness of bisimulation GKAT with UA. Despite bisimilarity being a less traditional equivalence in the context of Kleene algebra, the reduction provides a simpler completeness proof than the original one from [Smo+20] and

justifies the study of bisimilarity in the pursuit of completeness for GKAT. The section begins with an overview of the proof, and then proceeds with the formal details.

**Overview**

In a nutshell, the reduction of GKAT with UA to bisimulation GKAT with UA proceeds as follows: Given two expressions $e_1, e_2 \in GExp$ such that $\mathcal{L}(e_1) = \mathcal{L}(e_2)$, we can find expressions $\lfloor e_1 \rfloor, \lfloor e_2 \rfloor$ such that $e_i \equiv \lfloor e_i \rfloor$, and furthermore $\lfloor e_1 \rfloor \leftrightarrows \lfloor e_2 \rfloor$. Given completeness of bisimulation GKAT with UA, we obtain $e_1 \equiv \lfloor e_1 \rfloor \equiv_0 \lfloor e_2 \rfloor \equiv e_2$. Since $\equiv_0$ is contained in $\equiv$, $e_1 \equiv e_2$ by transitivity. Assuming we can construct the expressions $\lfloor e_1 \rfloor$ and $\lfloor e_2 \rfloor$, this line of reasoning proves the theorem below.

**Theorem 3.1.28** (Completeness II)**.** *For any $e, f \in GExp$, if $\mathcal{L}(e) = \mathcal{L}(f)$, then $e \equiv f$.*

The construction of $\lfloor e_1 \rfloor$ and $\lfloor e_2 \rfloor$ from $e_1$ and $e_2$ above relies on the uniqueness axiom and the characterization of bisimilarity equivalence classes of GKAT programs as behaviour trees (Section 3.1.3). Roughly, $\lfloor e_1 \rfloor$ and $\lfloor e_2 \rfloor$ are obtained by "pruning" redundant branches from the behaviour trees of $e_1$ and $e_2$, and translating the resulting behaviour trees back into expressions.

**Pruning dead branches**

Recall that the only difference between GKAT and bisimulation GKAT is the axiom (R0), which says that $e0 = 0$. Combining this observation with the tree analysis of behaviours of GKAT programs, it becomes clear that the guarded strings recognized by $e$ correspond precisely to the *live* branches of its behaviour tree, those that end in successful termination. The rest of the branches are *dead*, and are redundant with respect to the language semantics. The notions of *live* and *dead* generally apply to GKAT automata in the following way.

**Definition 3.1.29** ([Smo+20, Definition 5.3])**.** Given a GKAT automaton $(X, \delta_X)$, the set of *dead* states $D(X, \delta_X)$ is the largest subset $D \subseteq X$ such that for any $x \in D$ and $\alpha \in At$, either $x \downarrow \alpha$ or $x \xrightarrow{\alpha|p} y$ for some $y \in D$. A state that is not a dead state is *live*.

Equivalently, the set of live states in a GKAT automaton $(X, \delta_X)$ is the smallest subset $V \subseteq X$ such that (1) if $x \Rightarrow \alpha$ for some $\alpha \in At$, then $x \in V$, and (2) if $x \xrightarrow{\alpha|p} y$ for some $y \in V$, then $x \in V$.

Since expressions and trees form GKAT automata, they can be live or dead. The following lemma characterizes liveness of expressions as states of $(GExp, \delta)$.

**Lemma 3.1.30.** *Let $e \in GExp$. Then $e$ is dead if and only if $e \equiv_0 e0$.*

*Proof.* If $e \equiv_0 e0$, then $e \leftrightarrows e0$ by soundness. It follows that $e$ is dead, because $e0$ is dead. For the converse, we check that

$$R = \{(e, e0) \mid e \text{ is dead}\}$$

is a bisimulation by verifying the three properties in Lemma 3.1.16 and then conclude using the first completeness theorem of GKAT (Theorem 3.1.27).

1. We begin with the first property in Lemma 3.1.16, that $e \downarrow \alpha$ iff $e0 \downarrow \alpha$. If $e \downarrow \alpha$, then $e0 \downarrow \alpha$ by the transition rules. If $e0 \downarrow \alpha$, then either $e \downarrow \alpha$ or $e \Rightarrow \alpha$ and $0 \downarrow \alpha$. But $e$ is dead, so $\neg(e \Rightarrow \alpha)$. Thus, $e \downarrow \alpha$.
2. The second property is trivial: Since $e$ is dead, $\neg(e \Rightarrow \alpha)$ and $\neg(e0 \Rightarrow \alpha)$ for all $\alpha \in At$, so vacuously $e \Rightarrow \alpha$ iff $e0 \Rightarrow \alpha$.
3. Let $(e, e0) \in R$ and suppose $e \xrightarrow{\alpha|p} e'$ and $e0 \xrightarrow{\alpha|q} e''$. If $e'$ is live, then $e \xrightarrow{\alpha|p} e' \xrightarrow{\alpha_1|p_1} \cdots \xrightarrow{\alpha_n|p_n} f \Rightarrow \alpha$ for some $\alpha \in At$, implying that $e$ is live. Since $e$ is dead, this shows that $e'$ is dead as well. Furthermore, $e \xrightarrow{\alpha|p} e'$, $p = q$, and $e'' = e'0$ by the transition rules. This puts $(e', e'') \in R$, establishing the third property.

It follows that $R$ is a bisimulation. Since $(e, e0) \in R$, $e \leftrightarrows e0$, and by the first completeness theorem (Theorem 3.1.27), $e \equiv_0 e0$. □

Roughly, the expressions $\lfloor e_1 \rfloor$ and $\lfloor e_2 \rfloor$ appearing in the completeness proof sketch (see the text under the Overview heading above) are constructed by *pruning* dead subexpressions from $e_1$ and $e_2$. In Section 3.4, we will see how this can be done syntactically if $e_1$ and $e_2$ are of a particular form. In full generality, this is much easier to do at the level of the behaviour trees seen in Section 3.1.3. Recall that the set of all behaviour trees forms a final GKAT automaton $(Z, \zeta)$.

**Definition 3.1.31.** A *(behavioural) coequation* is a subset $C \subseteq Z$. A behaviour tree is called *nested* if it is of the form $!_\delta(e)$ for some $e \in GExp$. The *nesting coequation* is the set of all nested behaviour trees.

We define a *pruning operator* $\lfloor - \rfloor_C$ given $C \subseteq Z$ via *corecursion* [Rut98]. Informally, given a tree $t$, the tree $\lfloor t \rfloor_C$ is obtained by deleting every subtree of $t$ that

appears in $C$. This can be formally defined as follows: Consider the $G$-coalgebra $(Z, \zeta_C)$, where for any $t \in Z$, we define

$$\zeta_C(t)(\alpha) = \begin{cases} \bot & t(\alpha) \in \Sigma \text{ and } \partial_\alpha t \in C \\ (t(\alpha), \partial_\alpha t) & t(\alpha) \in \Sigma \text{ and } \partial_\alpha t \notin C \\ t(\alpha) & t(\alpha) \in 2 \end{cases}$$

The structure $\zeta_C$ is essentially obtained from $\zeta$ by deleting transitions to trees in $C$.

**Definition 3.1.32.** Given a coequation $C$, the *$C$-pruning operator* $\lfloor - \rfloor_C : Z \to Z$ is defined to be the behaviour map $!_{\zeta_C} : (Z, \zeta_C) \to (Z, \zeta)$, where $\zeta_C$ is defined above.

**Lemma 3.1.33.** *Given $C \subseteq Z$, the $C$-pruning map $\lfloor - \rfloor_C : Z \to Z$ satisfies the following for any $t \in Z$:*

$$\lfloor t \rfloor_C(\alpha) = \begin{cases} \bot & t(\alpha) \in \Sigma, \partial_\alpha t \in C \\ t(\alpha) & \textit{otherwise} \end{cases} \qquad \partial_\alpha \lfloor t \rfloor_C = \begin{cases} \lfloor \partial_\alpha t \rfloor_C & t(\alpha) \in \Sigma, \partial_\alpha t \notin C \\ \textit{undefined} & \textit{otherwise} \end{cases}$$

*Proof.* Both identities follow directly from Definition 3.1.32. □

The following surprising result tells us that the nesting coequation is stable under all pruning operators. Its proof is involved, so for the sake presentation I have given the proof its own Section 3.1.7.

**Theorem 3.1.34.** *For any coequation $C$, and for any nested $t$, $\lfloor t \rfloor_C$ is nested.*

*Proof.* See Section 3.1.7. □

In particular, if we take $D$ to be the coequation consisting of dead behaviour trees (the dead states of $(Z, \zeta)$), then $\lfloor !_\delta(e) \rfloor_D$ is nested, for any $e \in GExp$. That means there is an expression $\lfloor e \rfloor$ such that $!_\delta(\lfloor e \rfloor) = \lfloor !_\delta(e) \rfloor_D$. This expression is uniquely determined modulo bisimilarity, because $\lfloor e \rfloor \leftrightarroweq f$ implies that $!_\delta(\lfloor e \rfloor) = !_\delta(f)$ by Lemma 2.5.4. By the completeness theorem for bisimulation GKAT (Theorem 3.2.18), $\lfloor e \rfloor$ is uniquely determined up to $\equiv_0$.

**Definition 3.1.35.** The *syntax pruning operator* $\lfloor - \rfloor : GExp \to GExp/\equiv_0$ is defined $e \mapsto \lfloor e \rfloor$, where $\lfloor e \rfloor$ is the unique $\equiv_0$-class of $e$ such that $!_\delta(\lfloor e \rfloor) = \lfloor !_\delta(e) \rfloor_D$.

Note that we will often abuse notation and treat $\lfloor e \rfloor$ both as an expression—a representative of the $\equiv_0$-equivalence class $\lfloor e \rfloor$ denotes—and as a behaviour tree—the unique behaviour tree representing the $\equiv_0$-equivalence class of $e$ given by Theorem 3.1.27. The benefit of doing this is the same as the benefit of identifying solutions with the homomorphisms they represent (see the text after Lemma 3.1.26). Actually, syntax pruning can alternatively be defined as a certain GKAT automaton homomorphism, so long as we remove redundant transitions from $(GExp, \delta)$.

**Definition 3.1.36.** Let $(X, \delta_X)$ be a GKAT automaton. We define the *normalization* of $(X, \delta_X)$ to be the automaton $(X, \lfloor \delta_X \rfloor)$, where

$$\lfloor \delta_X \rfloor (x)(\alpha) = \begin{cases} \bot & \delta(x)(\alpha) = (p, x') \text{ and } x' \text{ is dead} \\ \delta_X(x)(\alpha) & \text{otherwise} \end{cases} \tag{3.7}$$

A GKAT automaton $(X, \delta_X)$ is called *normal* if $\delta_X = \lfloor \delta_X \rfloor$.

**Lemma 3.1.37.** *Syntax pruning is a* GKAT *automaton homomorphism*

$$\lfloor - \rfloor : (GExp, \lfloor \delta \rfloor) \to (GExp/\equiv_0, [\delta]_{\equiv_0})$$

*Proof.* We are going to show that the relation $R = \{(e, \lfloor e \rfloor) \mid e \in GExp\}$ is a bisimulation between $(GExp, \lfloor \delta \rfloor)$ and $(GExp/\equiv_0, [\delta]_{\equiv_0})$. To this end, let $e \in GExp$. If $e \downarrow_\delta \alpha$, then $e \downarrow_{\lfloor \delta \rfloor} \alpha$ and $\lfloor e \rfloor \downarrow \alpha$. Similarly if $e \Rightarrow_\delta \alpha$. If $e \xrightarrow{\alpha|p}_\delta e'$, there are two cases:

- If $e \downarrow_{\lfloor \delta \rfloor} \alpha$, then $e'$ is dead. This implies that $\lfloor e' \rfloor = \lfloor !_\delta(e') \rfloor_D$ is the constant $\bot$ behaviour tree, so $\lfloor e \rfloor (\alpha) = \bot$ and $\lfloor e \rfloor \downarrow \alpha$ as well.
- If $e \xrightarrow{\alpha|p}_{\lfloor \delta \rfloor} e'$, then $e'$ is live. This means that $!_\delta(e')$ is live as well, so $\lfloor e \rfloor \xrightarrow{\alpha|p} \lfloor e' \rfloor$ as desired (see Definition 3.1.32). □

The general goal of normalization is to semantically make language-equivalent states of a GKAT automaton bisimilar without changing the language they accept. This is the content of the following result.

**Lemma 3.1.38.** *Let $(X, \delta_X)$ be a* GKAT *automaton.*

*(i) For any $x \in X$, $\mathcal{L}(x, (X, \delta_X)) = \mathcal{L}(x, (X, \lfloor \delta_X \rfloor))$, and*

*(ii) for any $x, y \in X$, $x \leftrightarrows y$ in $(X, \lfloor \delta_X \rfloor)$ if and only if $\mathcal{L}(x) = \mathcal{L}(y)$.*

*Proof.* Item (i) is [Smo+20, Lemma 5.6(ii)]. Item (ii) is [Smo+20, Lemma 5.2]. □

In order for our proposed proof of Theorem 3.1.28 to go through, only two properties of the syntactic pruning operator need to be established.

**Theorem 3.1.39.** *For any $e \in GExp$, $e \equiv \lfloor e \rfloor$. Also, for any $e, f \in GExp$, $\mathcal{L}(e) = \mathcal{L}(f)$ if and only if $\lfloor e \rfloor \leftrightarrows \lfloor f \rfloor$.*

In order to prove Theorem 3.1.39, we need the following lemma.

**Lemma 3.1.40.** *If $\phi$ is a $\equiv_0$-solution to the left-affine system of equations $\mathcal{S}(X, \lfloor \delta_X \rfloor)$, then $\phi$ is also a $\equiv$-solution to the left-affine system $(X, \delta_X)$.*

*Proof.* Given $x \in X$, write the equation in $\mathcal{S}(X, \delta_X)$ associated with $x$ as

$$x = p_1 x_1 +_{b_1} \cdots +_{b_{n-1}} p_n x_n +_{b_n} c \qquad (x \text{ in } (X, \delta_X))$$

with $b_1, \ldots, b_n, c$ pairwise disjoint. Without loss of generality, we can assume that the dead states appearing in the equation above are $x_j, \ldots, x_n$. In which case, $\phi(x_j) \equiv_0 \cdots \equiv_0 \phi(x_n) \equiv_0 0$, because $\phi$ is a $\equiv_0$-solution to $(X, \lfloor \delta_X \rfloor)$ and $x_j, \ldots, x_n$ have no outgoing transitions. On the other hand, since in $(X, \lfloor \delta_X \rfloor)$ transitions are the same as in $(X, \delta_X)$ but without transitions to dead states, the equation in $\mathcal{S}(X, \lfloor \delta_X \rfloor)$ for $x$ is

$$x = p_1 x_1 +_{b_1} \cdots +_{b_{j-1}} (\bar{b}_j \cdots \bar{b}_n) c \qquad (x \text{ in } (X, \lfloor \delta_X \rfloor))$$

Putting these observations together,

$$\begin{aligned}
\phi(x) &\equiv_0 p_1 \phi(x_1) +_{b_1} \cdots +_{b_{j-1}} (\bar{b}_j \cdots \bar{b}_n) c && (\phi \equiv_0\text{-sol. to } (X, \lfloor \delta_X \rfloor)) \\
&\equiv_0 p_1 \phi(x_1) +_{b_1} \cdots +_{b_{j-1}} 0 +_{b_j} (\bar{b}_{j+1} \cdots \bar{b}_n) c && \text{(G9,G2,G3)} \\
&\equiv_0 p_1 \phi(x_1) +_{b_1} \cdots +_{b_{j-1}} 0 +_{b_j} \cdots +_{b_n} c && \text{(repeat prev.)} \\
&\equiv p_1 \phi(x_1) +_{b_1} \cdots +_{b_{j-1}} p_j \phi(x_j) +_{b_j} \cdots +_{b_n} c
\end{aligned}$$

(G9) is derived in Proposition 3.2.15 (see also Remark 3.2.16). This shows that $\phi$ is a $\equiv$-solution to $(X, \delta_X)$. □

*Proof of Theorem 3.1.39.* To see that (1) holds, observe that by Lemma 3.1.37, $\lfloor - \rfloor$ is a $\equiv_0$-solution to $(GExp, \lfloor \delta \rfloor)$. By Lemma 3.1.40, $\lfloor - \rfloor$ is also a $\equiv$-solution to $(GExp, \delta)$.

Now, since $(GExp, \delta)$ is locally finite, by UA it has a unique $\equiv$-solution. The identity is also a $\equiv$-solution to $(GExp, \delta)$ by Lemma 3.1.26, so it follows that $\lfloor - \rfloor \equiv \mathrm{id}$. In other words, $e \equiv \lfloor e \rfloor$ for all $e \in GExp$.

To see that (2) holds, let $\mathcal{L}(e) = \mathcal{L}(f)$. By (1) and soundness, $\mathcal{L}(e) = \mathcal{L}(\lfloor e \rfloor)$ and $\mathcal{L}(f) = \mathcal{L}(\lfloor f \rfloor)$. By Lemma 3.1.37, $e, (GExp, \lfloor \delta \rfloor) \leftrightarrows \lfloor e \rfloor, (GExp, \delta)$, and similarly for $f$. By Lemma 3.1.38, $e \leftrightarrows f$ in $(GExp, \lfloor \delta \rfloor)$, so by transitivity, $\lfloor e \rfloor \leftrightarrows \lfloor f \rfloor$. □

We are now ready to prove the main result of this section, Theorem 3.1.28, the completeness of GKAT with UA for language equivalence.

*Proof of Theorem 3.1.28.* Let $e_1, e_2 \in GExp$ such that $\mathcal{L}(e_1) = \mathcal{L}(e_2)$. By Theorem 3.1.39, we can find expressions $\lfloor e_1 \rfloor, \lfloor e_2 \rfloor$ such that $e_i \equiv \lfloor e_i \rfloor$ and $\lfloor e_1 \rfloor \leftrightarrows \lfloor e_2 \rfloor$. Given completeness of bisimulation GKAT with UA (Theorem 3.1.27), we obtain $e_1 \equiv \lfloor e_1 \rfloor \equiv_0 \lfloor e_2 \rfloor \equiv e_2$. Since $\equiv_0$ is contained in $\equiv$, $e_1 \equiv e_2$ by transitivity. □

### 3.1.7 The Proof of Theorem 3.1.34

This subsection is dedicated to the proof of Theorem 3.1.34, which requires a particular construction on behaviour trees. Namely, we need to be able to compose behaviour trees using the algebraic operations of GKAT. Corecursively, given $s, t \in Z$, $b \in BExp$, we define

$$(s +_b t)(\alpha) = \begin{cases} s(\alpha) & \alpha \leq b \\ t(\alpha) & \alpha \leq \bar{b} \end{cases} \qquad \partial_\alpha(s +_b t) = \begin{cases} \partial_\alpha s & \alpha \leq b \\ \partial_\alpha t & \alpha \leq \bar{b} \end{cases}$$

$$(st)(\alpha) = \begin{cases} s(\alpha) & s(\alpha) \in \bot + \Sigma \\ t(\alpha) & s(\alpha) = \top \end{cases} \qquad \partial_\alpha(st) = \begin{cases} (\partial_\alpha s)t & s(\alpha) \in \Sigma \\ \partial_\alpha t & s(\alpha) = \top \end{cases}$$

$$s^{(b)}(\alpha) = \begin{cases} s(\alpha) & s(\alpha) \in \Sigma, \alpha \leq b \\ \top & \alpha \leq \bar{b} \\ \bot & \text{otherwise} \end{cases} \qquad \partial_\alpha s^{(b)} = \begin{cases} (\partial_\alpha s) s^{(b)} & s(\alpha) \in \Sigma, \alpha \leq b \\ \text{undefined} & \text{otherwise} \end{cases}$$

This extends all of the GKAT programming constructs to behaviour trees. We can now define the map $[\![ - ]\!] : GExp \to Z$, the *initial semantics of bisimulation* GKAT,

recursively on GKAT expressions:

$$[\![b]\!] = !_\delta(b) \qquad [\![p]\!] = !_\delta(p) \qquad [\![e +_b f]\!] = [\![e]\!] +_b [\![f]\!] \tag{3.8}$$

$$[\![ef]\!] = [\![e]\!]\,[\![f]\!] \qquad \left[\!\left[e^{(b)}\right]\!\right] = [\![e]\!]^{(b)} \tag{3.9}$$

for any $b \in BExp$ and $p \in \Sigma$, $e, f \in GExp$.

Recall that the bisimulation semantics of GKAT is given by the unique coalgebra homomorphism $!_\delta : (GExp, \delta) \to (Z, \zeta)$ (Lemma 2.5.4 and Theorem 3.1.19). The next theorem says that the initial semantics coincides with the bisimulation semantics.

**Theorem 3.1.41.** *For any $e \in GExp$, $[\![e]\!] = !_\delta(e)$.*

*Proof.* It suffices to show that $[\![-]\!]$ is a GKAT automaton homomorphism. This amounts to showing that the following rules hold:

$$\frac{e \downarrow \alpha}{[\![e]\!](\alpha) = 0} \qquad \frac{e \Rightarrow \alpha}{[\![e]\!](\alpha) = 1} \qquad \frac{e \xrightarrow{\alpha|p} e'}{[\![e]\!] \xrightarrow{\alpha|p} [\![e']\!]}$$

We do this by induction on the transition rules for $e$. In the base case, there are two subcases.

- By definition, $[\![b]\!](\alpha) = 0$ if and only if $b \downarrow \alpha$, and $[\![b]\!](\alpha) = 1$ if and only if $b \Rightarrow \alpha$. Since $b$ does not admit any transitions in $(GExp, \delta)$, the last implication holds vacuously.
- We have that $p \xrightarrow{\alpha|p} 1$ for any $\alpha \in At$. By definition of $[\![p]\!]$, we have $[\![p]\!](\alpha) = p$ and $\partial_\alpha [\![p]\!] = [\![1]\!]$, and hence $[\![p]\!] \xrightarrow{\alpha|p} [\![1]\!]$. Furthermore, $p$ does not terminate (successfully or unsuccessfully) in $(GExp, \delta)$, so the first two rules hold vacuously.

In the inductive step, suppose the inferences above hold for $e$ and $f$, and $b \in BExp$.

- If $e +_b f \downarrow \alpha$, then either $\alpha \leq b$ and $e \downarrow \alpha$, or $\alpha \leq \bar{b}$ and $f \downarrow \alpha$. In the first case, $[\![e +_b f]\!](\alpha) = [\![e]\!](\alpha) = 0$, and in the second $[\![e +_b f]\!](\alpha) = [\![f]\!](\alpha) = 0$. Furthermore, if $e +_b f \Rightarrow \alpha$, then either $\alpha \leq b$ and $e \Rightarrow \alpha$, or $\alpha \leq \bar{b}$ and $f \Rightarrow \alpha$. In the first case, $[\![e +_b f]\!](\alpha) = [\![e]\!](\alpha) = 1$, and in the second $[\![e +_b f]\!](\alpha) = [\![f]\!](\alpha) = 1$. Finally, if $e +_b f \xrightarrow{\alpha|p} g$, then either $\alpha \leq b$ and $e \xrightarrow{\alpha|p} g$, or $\alpha \leq \bar{b}$ and $f \xrightarrow{\alpha|p} g$. In the first case, $[\![e +_b f]\!](\alpha) = [\![e]\!](\alpha) = p$ and $\partial_\alpha [\![e +_b f]\!] =$

$\partial_\alpha(\llbracket e \rrbracket +_b \llbracket f \rrbracket) = \partial_\alpha \llbracket e \rrbracket = \llbracket g \rrbracket$, and in the second, $\llbracket e +_b f \rrbracket(\alpha) = \llbracket f \rrbracket(\alpha) = p$ and $\partial_\alpha \llbracket e +_b f \rrbracket = \partial_\alpha \llbracket f \rrbracket = \llbracket g \rrbracket$.

- If $ef \downarrow \alpha$, then either $e \downarrow \alpha$, or $e \Rightarrow \alpha$ and $f \downarrow \alpha$. In the first case, $\llbracket e \rrbracket(\alpha) = 0$ and $\llbracket ef \rrbracket(\alpha) = \llbracket e \rrbracket(\alpha) = 0$, and in the second, $\llbracket ef \rrbracket(\alpha) = \llbracket e \rrbracket \llbracket f \rrbracket(\alpha) = \llbracket f \rrbracket(\alpha) = 0$. Furthermore, if $ef \Rightarrow \alpha$, then $e \Rightarrow \alpha$ and $f \Rightarrow \alpha$. Thus, $\llbracket ef \rrbracket(\alpha) = \llbracket f \rrbracket(\alpha) = 1$. Finally, if $ef \xrightarrow{\alpha|p} g$, then either $e \Rightarrow \alpha$ and $f \xrightarrow{\alpha|p} g$, or $e \xrightarrow{\alpha|p} e'$ and $g = e'f$. In the first case, $\llbracket ef \rrbracket(\alpha) = \llbracket f \rrbracket(\alpha) = p$ and

$$\partial_\alpha \llbracket ef \rrbracket = \partial_\alpha(\llbracket e \rrbracket \llbracket f \rrbracket) = \partial_\alpha \llbracket f \rrbracket = \llbracket g \rrbracket,$$

meaning $\llbracket ef \rrbracket \xrightarrow{\alpha|p} \llbracket g \rrbracket$, and in the second $\llbracket ef \rrbracket(\alpha) = \llbracket e \rrbracket(\alpha) = p$, and

$$\partial_\alpha \llbracket ef \rrbracket = \partial_\alpha \llbracket e \rrbracket \llbracket f \rrbracket = \llbracket e' \rrbracket \llbracket f \rrbracket = \llbracket g \rrbracket,$$

thus showing that $\llbracket ef \rrbracket \xrightarrow{\alpha|p} \llbracket g \rrbracket$ again.

- If $e^{(b)} \downarrow \alpha$, then $\alpha \leq b$ and either $e \downarrow \alpha$ or $e \Rightarrow \alpha$. In either case, $\llbracket e^{(b)} \rrbracket(\alpha) = \llbracket e \rrbracket^{(b)}(\alpha) = 0$. Furthermore, if $e^{(b)} \Rightarrow \alpha$, then $\alpha \leq \bar{b}$ and $\llbracket e^{(b)} \rrbracket = \llbracket e \rrbracket^{(b)}(\alpha) = 1$ by definition. Finally, if $e^{(b)} \xrightarrow{\alpha|p} g$, then $\alpha \leq b$, $e \xrightarrow{\alpha|p} e'$, and $g = e'e^{(b)}$. This means that $\llbracket e^{(b)} \rrbracket(\alpha) = \llbracket e \rrbracket^{(b)}(\alpha) = \llbracket e \rrbracket(\alpha) = p$ and

$$\partial_\alpha \left\llbracket e^{(b)} \right\rrbracket = \partial_\alpha \llbracket e \rrbracket^{(b)} = (\partial_\alpha \llbracket e \rrbracket) \llbracket e \rrbracket^{(b)} = \llbracket e' \rrbracket \left\llbracket e^{(b)} \right\rrbracket = \llbracket g \rrbracket. \qquad \square$$

Next, we are going to show that pruning interacts compositionally with the algebraic operations on trees. In particular, given $s,t \in Z$ and $b \in BExp$, we show that the $C$-prunings of $st$ and $s^{(b)}$ are prunings of $s$ with respect to different coequations.

**Lemma 3.1.42.** *For any $s,t \in Z$, $b \in BExp$, and $C \subseteq Z$,*

1. $\lfloor st \rfloor_C = \lfloor s \rfloor_{C^\dagger}$ *where* $C^\dagger = \{r \mid rt \in C\}$.
2. $\left\lfloor s^{(b)} \right\rfloor_C = (\lfloor s \rfloor_{C^\bullet})^{(b)}$ *where* $C^\bullet = \{r \mid rs^{(b)} \in C\}$.

In the proof of this lemma, we use Lemma 3.1.16 as it applies to $(Z,\zeta)$. Namely, that if for any $(s,t) \in R$ and $\alpha \in At$, $s(\alpha) = t(\alpha)$ and $(\partial_\alpha s, \partial_\alpha t) \in R$, then $R$ is a bisimulation on $(Z,\zeta)$.

*Proof.* We are going to show that the following two relations are bisimulations.

$$R = \{(\lfloor st \rfloor_C, \lfloor s \rfloor_{C^\dagger} \lfloor t \rfloor_C) \mid s,t \in Z\} \cup \Delta_Z \tag{3.10}$$

$$Q = \left\{ (r\lfloor s^{(b)} \rfloor_C, r(\lfloor s \rfloor_{C^\bullet})^{(b)}) \;\middle|\; r,s \in Z \right\} \cup \Delta_Z \tag{3.11}$$

where $\Delta_Z = \{(s,s) \mid s \in Z\}$. To see that $R$ is a bisimulation, let $\alpha \in At$ and observe

$$\lfloor st \rfloor_C(\alpha) = \begin{cases} \bot & \partial_\alpha(st) \in C \\ st(\alpha) & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_\_\text{)}$$

$$= \begin{cases} \bot & s(\alpha) \in \Sigma, \partial_\alpha s \in C^\dagger \\ \bot & s(\alpha) = \top, \partial_\alpha t \in C \\ s(\alpha) & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\dagger \\ t(\alpha) & s(\alpha) = \top, \partial_\alpha t \notin C \end{cases} \qquad \left( \begin{array}{c} \text{expanding the} \\ \text{conditions} \end{array} \right)$$

$$= \begin{cases} \bot & \lfloor s \rfloor_{C^\dagger}(\alpha) = \bot \\ \bot & s(\alpha) = \top, \lfloor t \rfloor_C(\alpha) = \bot \\ \lfloor s \rfloor_{C^\dagger}(\alpha) & \lfloor s \rfloor_{C^\dagger}(\alpha) \in \Sigma \\ \lfloor t \rfloor_C(\alpha) & \lfloor s \rfloor_{C^\dagger}(\alpha) = \top \end{cases} \qquad \text{(def. } \lfloor - \rfloor_\_\text{)}$$

$$= \lfloor s \rfloor_{C^\dagger} \lfloor t \rfloor_C(\alpha)$$

On the other hand, we have

$$\partial_\alpha \lfloor st \rfloor_C = \begin{cases} \lfloor \partial_\alpha(st) \rfloor_C & st(\alpha) \in \Sigma, \partial_\alpha(st) \notin C \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_C\text{)}$$

$$= \begin{cases} \lfloor (\partial_\alpha s)t \rfloor_C & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\dagger \\ \lfloor \partial_\alpha t \rfloor_C & s(\alpha) = \top, \partial_\alpha t \notin C \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \partial_\alpha(st), C^\dagger\text{)}$$

$$\partial_\alpha(\lfloor s \rfloor_{C^\dagger} \lfloor t \rfloor_C) = \begin{cases} (\partial_\alpha \lfloor s \rfloor_{C^\dagger}) \lfloor t \rfloor_C & \lfloor s \rfloor_{C^\dagger} \in \Sigma \\ \partial_\alpha \lfloor t \rfloor_C & \lfloor s \rfloor_{C^\dagger}(\alpha) = \top \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \partial_\alpha(\lfloor s \rfloor_{C^\dagger} \lfloor t \rfloor_C)\text{)}$$

$$= \begin{cases} (\partial_\alpha \lfloor s \rfloor_{C^\dagger}) \lfloor t \rfloor_C & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\dagger \\ \lfloor \partial_\alpha t \rfloor_C & s(\alpha) = \top, \partial_\alpha t \notin C \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_- \text{)}$$

$$= \begin{cases} \lfloor \partial_\alpha s \rfloor_{C^\dagger} \lfloor t \rfloor_C & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\dagger \\ \lfloor \partial_\alpha t \rfloor_C & s(\alpha) = \top, \partial_\alpha t \notin C \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_- \text{)}$$

It follows that $(\partial_\alpha \lfloor st \rfloor_C, \partial_\alpha(\lfloor s \rfloor_{C^\dagger} \lfloor t \rfloor_C)) \in R$. Hence, $R$ is a bisimulation. By finality of $(Z, \zeta)$, $R \subseteq \Delta_Z$, so $\lfloor st \rfloor_C = \lfloor s \rfloor_{C^\dagger}$. Now consider $Q$. If $r(\alpha) \in \bot + \Sigma$, then

$$r \left\lfloor s^{(b)} \right\rfloor_C (\alpha) = r(\alpha) = r(\lfloor s \rfloor_{C^\bullet})^{(b)}(\alpha)$$
$$\partial_\alpha(r \left\lfloor s^{(b)} \right\rfloor_C) = (\partial_\alpha r) \left\lfloor s^{(b)} \right\rfloor_C$$
$$\partial_\alpha(r(\lfloor s \rfloor_{C^\bullet})^{(b)}) = (\partial_\alpha r)(\lfloor s \rfloor_{C^\bullet})^{(b)}$$

Hence, it suffices to consider the case in which $r(\alpha) = \top$. For $\alpha \leq \bar{b}$ in this case, $r \left\lfloor s^{(b)} \right\rfloor_C (\alpha) = \top = r \lfloor s \rfloor^{(b)}{}_{C^\bullet}(\alpha)$. For $\alpha \leq b$ in this case,

$$r\lfloor s^{(b)} \rfloor_C(\alpha) = \begin{cases} \bot & s^{(b)}(\alpha) \in \Sigma, \partial_\alpha(s^{(b)}) \in C \\ s^{(b)}(\alpha) & \text{otherwise} \end{cases}$$

$$= \begin{cases} \bot & s(\alpha) \in \Sigma, (\partial_\alpha s) s^{(b)} \in C \\ s^{(b)}(\alpha) & \text{otherwise} \end{cases} \qquad \text{(def. } s^{(b)} \text{)}$$

$$= \begin{cases} \bot & s(\alpha) \in \Sigma, \partial_\alpha s \in C^\bullet \\ s^{(b)}(\alpha) & \text{otherwise} \end{cases} \qquad \text{(def. } C^\bullet \text{)}$$

$$= \begin{cases} \bot & s(\alpha) \in \Sigma, \partial_\alpha s \in C^\bullet \\ \bot & s(\alpha) = \top \\ s(\alpha) & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\bullet \end{cases} \qquad \text{(def. } s^{(b)} \text{)}$$

$$= \begin{cases} \bot & \lfloor s \rfloor_{C^\bullet}(\alpha) = \bot \\ \bot & \lfloor s \rfloor_{C^\bullet}(\alpha) = \top \\ \lfloor s \rfloor_{C^\bullet}(\alpha) & \lfloor s \rfloor_{C^\bullet}(\alpha) \in \Sigma \end{cases} \qquad \text{(def. } \lfloor - \rfloor_- \text{)}$$

$$= r(\lfloor s \rfloor_{C^\bullet})^{(b)}(\alpha)$$

on the one hand, and on the other we have both

$$\partial_\alpha(r\lfloor s^{(b)} \rfloor_C) = \begin{cases} \lfloor \partial_\alpha s^{(b)} \rfloor_C & s^{(b)}(\alpha) \in \Sigma, \partial_\alpha s^{(b)} \notin C \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$= \begin{cases} \lfloor (\partial_\alpha s) s^{(b)} \rfloor_C & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\bullet \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } s^{(b)}, C^\bullet \text{)}$$

$$= \begin{cases} \lfloor \partial_\alpha s \rfloor_{C^\bullet} \lfloor s^{(b)} \rfloor_C & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\bullet \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(1., with } t = s^{(b)} \text{)}$$

$$\partial_\alpha(r(\lfloor s \rfloor_{C^\bullet})^{(b)}) = \begin{cases} (\partial_\alpha \lfloor s \rfloor_{C^\bullet})(\lfloor s \rfloor_{C^\bullet})^{(b)} & \lfloor s \rfloor_{C^\bullet}(\alpha) \in \Sigma \\ \text{undefined} & \text{otherwise} \end{cases}$$

$$= \begin{cases} (\partial_\alpha \lfloor s \rfloor_{C^\bullet})(\lfloor s \rfloor_{C^\bullet})^{(b)} & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\bullet \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_- \text{)}$$

$$= \begin{cases} \lfloor \partial_\alpha s \rfloor_{C^\bullet} (\lfloor s \rfloor_{C^\bullet})^{(b)} & s(\alpha) \in \Sigma, \partial_\alpha s \notin C^\bullet \\ \text{undefined} & \text{otherwise} \end{cases} \qquad \text{(def. } \lfloor - \rfloor_- \text{)}$$

Hence, $(\partial_\alpha(r\lfloor s^{(b)} \rfloor_C), \partial_\alpha(r(\lfloor s \rfloor_{C^\bullet})^{(b)})) \in Q$. This shows that $Q$ is a bisimulation, so $\lfloor s^{(b)} \rfloor_C = (\lfloor s \rfloor_{C^\bullet})^{(b)}$ as desired. □

We are now ready to prove Theorem 3.1.34, the focus of this subsection.

*Proof of Theorem 3.1.34.* Let $t = !_\delta(e)$. We proceed by induction on $e$. In the base cases, $t = !_\delta(b)$ for some $b \in BExp$ or $t = !_\delta(p)$ for some $p \in \Sigma$. In the first base case, since $\mathrm{dom}(!_\delta(b)) \subseteq At$, $\lfloor !_\delta(b) \rfloor_C = !_\delta(b)$, by definition of $\lfloor t \rfloor_C$. In the second base case, if $t = !_\delta(p)$, then there are two possibilities to consider. If $!_\delta(1) \in C$, then $\partial_\alpha t \in C$ and $\lfloor t \rfloor_C = !_\delta(p0)$. If $!_\delta(1) \notin C$, then $\lfloor t \rfloor_C = !_\delta(p)$.

For the inductive step, there are three cases.

- Suppose $t = !_\delta(e_1 +_b e_2)$. Let $f_i \in GExp$ satisfy $!_\delta(f_i) = \lfloor !_\delta(e_i) \rfloor_C$. Then $\lfloor t \rfloor_C = !_\delta(f_1 +_b f_2)$. Indeed, if $\alpha \leq b$, then

$$\lfloor t \rfloor_C(\alpha) = \begin{cases} \bot & \partial_\alpha t \in C \\ t(\alpha) & \text{otherwise} \end{cases} \qquad \partial_\alpha \lfloor t \rfloor_C = \begin{cases} \text{undefined} & \partial_\alpha t \in C \\ \lfloor \partial_\alpha t \rfloor_C & \text{otherwise} \end{cases}$$

$$= \begin{cases} \bot & \partial_\alpha !_\delta(e_1) \in C \\ !_\delta(e_1)(\alpha) & \text{otherwise} \end{cases} \qquad = \begin{cases} \text{undefined} & \partial_\alpha !_\delta(e_1) \in C \\ \lfloor \partial_\alpha !_\delta(e_1) \rfloor_C & \text{otherwise} \end{cases}$$

$$= !_\delta(f_1)(\alpha) \qquad = \partial_\alpha !_\delta(f_1)$$

  and similarly if $\alpha \leq \bar{b}$. It follows that $\lfloor t \rfloor_C = !_\delta(f_1 +_b f_2)$.

- Suppose $t = !_\delta(e_1 e_2)$ and using the induction hypothesis let $!_\delta(f_1) = \lfloor !_\delta(e_1) \rfloor_{C^\dagger}$ and $!_\delta(f_2) = \lfloor !_\delta(e_2) \rfloor_C$, where $C^\dagger = \{!_\delta(g) \mid !(ge_2) \in C\}$. We show that $\lfloor t \rfloor_C = !_\delta(f_1 f_2)$ as follows. By Theorem 3.1.41, $C^\dagger = \{!_\delta(g) \mid !_\delta(g)!_\delta(e_2) \in C\}$, so

$$\begin{aligned} !_\delta(f_1 f_2) &= !_\delta(f_1)!_\delta(f_2) && \text{(Theorem 3.1.41)} \\ &= \lfloor !_\delta(e_1) \rfloor_{C^\dagger} \lfloor !_\delta(e_2) \rfloor_C && \text{(ind. hyp.)} \\ &= \lfloor !_\delta(e_1)!_\delta(e_2) \rfloor_C && \text{(Lemma 3.1.42)} \\ &= \lfloor !_\delta(e_1 e_2) \rfloor_C && \text{(Theorem 3.1.41)} \\ &= \lfloor t \rfloor_C \end{aligned}$$

- Suppose $t = !_\delta(e^{(b)})$ and using the induction hypothesis let $!_\delta(f) = \lfloor !_\delta(e) \rfloor_{C^\bullet}$ where $C^\bullet = \{!_\delta(g) \mid !_\delta(ge^{(b)}) \in C\}$. We are going to show that $\lfloor t \rfloor_C = !_\delta(f^{(b)})$. By Theorem 3.1.41, $C^\bullet = \{!_\delta(g) \mid !_\delta(g)!_\delta(e)^{(b)} \in C\}$, like in Lemma 3.1.42, and

$$\begin{aligned} !_\delta(f^{(b)}) &= !_\delta(f)^{(b)} && \text{(Theorem 3.1.41)} \\ &= \lfloor !_\delta(e) \rfloor_{C^\bullet}{}^{(b)} && \text{(ind. hyp.)} \\ &= \left\lfloor !_\delta(e)^{(b)} \right\rfloor_C && \text{(Lemma 3.1.42)} \\ &= \left\lfloor !_\delta(e^{(b)}) \right\rfloor_C && \text{(Theorem 3.1.41)} \\ &= \lfloor t \rfloor_C && \square \end{aligned}$$

### 3.1.8 Concluding remarks about GKAT

In this section, we saw an overview of the language and axiomatization of GKAT [Smo+20], an inference system for deriving equivalences between simple imperative programs built from if-then-else branching and while loops. The axiomatization is not ideal for two reasons: The axiomatization of recursion, i.e., (RSP$^*$) and UA, are non-algebraic, meaning they are not sound under substitution of action symbols for terms. In particular, UA is cumbersome for algebraic reasoning, as it requires the construction of a solvable left-affine system in every application. This situation is an instance of a far more general axiomatization problem, as we will see in Chapter 4. In the next section, we address non-algebraicity and elimination of UA from the axiomatization by restricting our attention to a fragment of GKAT.

## 3.2 Skip-free Guarded Kleene Algebra with Tests

In this section, we introduce what we call the skip-free fragment of GKAT, consisting of programs that do not contain assert statements in the body (other than assert false). In other words, Boolean statements are restricted to control statements, formally only appearing in conditional $+_b$ and loop constructs. For this fragment, we show that the axiom scheme UA can be avoided entirely. In fact, this is true for language semantics as well as for bisimulation semantics.

More specifically, we prove two completeness results: In Section 3.3, we start by focusing on the *bisimulation semantics* of the skip-free fragment, and then in Section 3.4 expand our argument to its *language semantics*. We first provide a reduction of the completeness of skip-free GKAT *modulo bisimilarity* to the completeness of Grabmayer and Fokkink's one-free regular expressions modulo bisimilarity, the case study in Chapter 2 that culminated in Theorem 2.4.13. We then provide a reduction of the completeness of skip-free GKAT modulo language semantics to the completeness of skip-free GKAT modulo bisimilarity via a technique inspired by the *tree pruning* approach of [Sch+21], which appeared in Section 3.1.6.

Finally, in Section 3.5, we connect our semantics of skip-free GKAT expressions to the established semantics of full GKAT. We also connect the syntactic proofs between skip-free GKAT expressions in both our axiomatization and the existing one. In conjunction with the results of Sections 3.3 and 3.4, the results in Section 3.5 are

a significant step towards determining whether the axioms of GKAT give a complete description of program equivalence.

### 3.2.1 Skip-free Expressions

The fragment of GKAT in focus is the one that *excludes sub-programs that can accept immediately* without performing any action. Since the excluded programs necessarily involve the "skip" command, we call the fragment that avoids them *skip-free*. Among others, programs of the form assert b for $b \neq$ false and while false do $p$, which is equivalent to assert true, are not skip-free GKAT programs.

**Definition 3.2.1.** Given a set $\Sigma$ of *atomic actions*, the set $GExp_{\mathsf{sf}}$ of *skip-free* GKAT *expressions* is given by the grammar

$$GExp_{\mathsf{sf}} \ni e_1, e_2 ::= 0 \mid p \in \Sigma \mid e_1 +_b e_2 \mid e_1 \cdot e_2 \mid {e_1}^{(b)} e_2$$

where $b$ ranges over the Boolean algebra expressions $BExp$.

Unlike full GKAT, in skip-free GKAT the loop construct is treated as a binary operation, analogous to Kleene's original star operation [Kle56], which was also binary. This helps us avoid loops of the form $e^{(b)}$, which skips when $b$ does not hold. The expression ${e_1}^{(b)} e_2$ corresponds to ${e_1}^{(b)} \cdot e_2$ in GKAT.

*Example* 3.2.2. Using the same notational shorthand as in Example 3.1.1, the block of code representing fizzbuzz2 in Figure 3.1 can be cast as the skip-free GKAT expression

$$(n := 1) \cdot ((fizzbuzz +_{3|n \wedge 5|n} (fizz +_{3|n} (buzz +_{5|n} n))) \cdot n{+}{+})^{(n \leq 100)} (done!)$$

Note how we use a skip-free loop of the form ${e_1}^{(b)} e_2$ instead of the looping construct ${e_1}^{(b)}$ before concatenating with $e_2$, as was done for GKAT.

*Remark* 3.2.3. As it is missing the skip command, skip-free GKAT is automatically strictly less expressive than GKAT. However, one might wonder if adding a simple *done!* action to the end of a GKAT program creates a skip-free GKAT program. This is false: For instance, if $0 < b < 1$, $p, q \in \Sigma$, then $(q(p +_b 1))^{(b)} p$ is not expressible in skip-free GKAT. This statement will be made more precise in Example 3.5.5.

$$\frac{}{p \xrightarrow{\alpha|p} \checkmark} \qquad \frac{e_1 \xrightarrow{\alpha|p} \xi \quad \alpha \leq b}{e_1 +_b e_2 \xrightarrow{\alpha|p} \xi} \qquad \frac{e_2 \xrightarrow{\alpha|p} \xi \quad \alpha \leq \bar{b}}{e_1 +_b e_2 \xrightarrow{\alpha|p} \xi} \qquad \frac{e_1 \xrightarrow{\alpha|p} e'}{e_1 e_2 \xrightarrow{\alpha|p} e' e_2}$$

$$\frac{e_1 \xrightarrow{\alpha|p} \checkmark}{e_1 e_2 \xrightarrow{\alpha|p} e_2} \qquad \frac{e_1 \xrightarrow{\alpha|p} e' \quad \alpha \leq b}{e_1{}^{(b)} e_2 \xrightarrow{\alpha|p} e'(e_1{}^{(b)} e_2)} \qquad \frac{e_1 \xrightarrow{\alpha|p} \checkmark \quad \alpha \leq b}{e_1{}^{(b)} e_2 \xrightarrow{\alpha|p} e_1{}^{(b)} e_2} \qquad \frac{e_2 \xrightarrow{\alpha|p} \xi \quad \alpha \leq \bar{b}}{e_1{}^{(b)} e_2 \xrightarrow{\alpha|p} \xi}$$

**Figure 3.6:** The small-step semantics of skip-free GKAT expressions, $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. Above, $e_1, e_2, e' \in GExp_{\mathsf{sf}}$, $p \in \Sigma$, $b \in BExp$, $\alpha \in At$, and $\xi \in \checkmark + GExp_{\mathsf{sf}}$.

### 3.2.2 Semantics

Like GKAT, we interpret skip-free GKAT expressions as *automata* (small-step semantics), as *behaviours* (operational semantics), and as *languages* (language semantics).[8]

#### Automata

The small-step semantics of skip-free GKAT uses a slight variation on GKAT automata.

**Definition 3.2.4.** A *skip-free automaton* is an $H$-coalgebra $(X, \delta_X)$, where the functor $H$ is defined for any $X$ and $h\colon X \to Y$ by

$$HX = (\bot + \Sigma \times (\checkmark + X))^{At} \qquad H(h)(\theta)(\alpha) = \begin{cases} \bot & \theta(\alpha) = \bot \\ (p, \checkmark) & \theta(\alpha) = (p, \checkmark) \\ (p, h(x)) & \theta(\alpha) = (p, x) \in \Sigma \times X \end{cases}$$

for $\theta \in HX$ and $\alpha \in At$. Like in a GKAT automaton, we write $x \xrightarrow{\alpha|p} \xi$ if $\delta_X(x)(\alpha) = (p, \xi)$, although $\xi$ can be the successful termination symbol $\checkmark$ in skip-free automata. We write $x \downarrow \alpha$ if $\delta_X(x)(\alpha) = \bot$. We also adopt the convention of writing $x \xrightarrow{b|p} x'$ where $b \in BExp$ to represent the set of transitions $x \xrightarrow{\alpha|p} x'$ with $\alpha \leq b$.

**Definition 3.2.5.** We equip the set $GExp_{\mathsf{sf}}$ of all skip-free GKAT expressions with the automaton structure $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ given in Figure 3.6, representing step-by-step execution. Given $e \in GExp_{\mathsf{sf}}$, we denote the set of states reachable from $e$ by $\langle e \rangle$ and call this the *small-step semantics* of $e$.

The small-step semantics of skip-free GKAT expressions is inspired by Brzozowski's *derivatives* [Brz64], which provide an automata-theoretic description of the

[8] We will connect these to the relational semantics from Definition 3.1.3 in Section 3.5.

step-by-step execution of a regular expression. Our first lemma tells us that, like one-free regular expressions (Lemma 2.4.5) and GKAT expressions (Lemma 3.1.14), skip-free GKAT expressions correspond to finite automata.

**Lemma 3.2.6.** *For any $e \in GExp_{\mathsf{sf}}$, $\langle e \rangle$ has finitely many states.*

*Proof.* By induction on $e$. The automaton $\langle 0 \rangle$ has a single state with no transitions and $\langle p \rangle$ consists of a single state that accepts all of $At$ after $p$. Write $\#(e)$ for the number of expressions reachable from $e$. For the inductive step, it suffices to notice that $\#(e_1 +_b e_2)$, $\#(e_1 e_2)$, and $\#(e_1 {}^{(b)} e_2)$ are bounded above by $\#(e_1) + \#(e_2) + 1$. This is clear in all cases except for $e_1 {}^{(b)} e_2$. In the $e_1 {}^{(b)} e_2$ case, observe that every state in $\langle e_1 {}^{(b)} e_2 \rangle$ (other than $e_1 {}^{(b)} e_2$) is either of the form $e_1' (e_1 {}^{(b)} e_2)$ for some $e_1' \in \langle e_1 \rangle$ or is of the form $e_2'$ for some state $e_2' \in \langle e_2 \rangle$. These are in one-to-one correspondence with elements of $\langle e_1 \rangle \sqcup \langle e_2 \rangle$, so $\#(e_1 {}^{(b)} e_2) \leq \#(e_1) + \#(e_2) + 1$. □

*Remark* 3.2.7. Another way to formulate the transition structure of $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ is to define $\delta_{\mathsf{sf}}$ directly. Given $e_1, e_2 \in GExp$, $p \in \Sigma$, and $b \in BExp$, we set

$$\delta_{\mathsf{sf}}(0)(\alpha) = \bot \qquad \delta_{\mathsf{sf}}(p)(\alpha) = (p, \checkmark) \qquad \delta_{\mathsf{sf}}(e_1 +_b e_2)(\alpha) = \begin{cases} \delta_{\mathsf{sf}}(e_1)(\alpha) & \alpha \leq b \\ \delta_{\mathsf{sf}}(e_2)(\alpha) & \alpha \leq \bar{b} \end{cases}$$

$$\delta_{\mathsf{sf}}(e_1 e_2)(\alpha) = \begin{cases} (p, e' e_2) & \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e') \\ (p, e_2) & \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, \checkmark) \\ \bot & \delta_{\mathsf{sf}}(e_1)(\alpha) = \bot \end{cases}$$

$$\delta_{\mathsf{sf}}(e_1 {}^{(b)} e_2) = \begin{cases} (p, e'(e_1 {}^{(b)} e_2)) & \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e') \\ (p, e_1 {}^{(b)} e_2) & \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, \checkmark) \\ \bot & \delta_{\mathsf{sf}}(e_1)(\alpha) = \bot \end{cases}$$

*Example* 3.2.8. The automaton that arises from the program fizzbuzz2 is depicted in Figure 3.7, with $b = n \leq 100$, $c = 3|n$, and $d = 5|n$. The expression $e$ is the same as in Example 3.2.2, $e_1$ is the $e$ but without $n := 0$ in front, and $e_2 = (n{+}{+}) \cdot e_1$.

The automaton interpretation of a skip-free GKAT expression (its small-step semantics) provides an intuitive visual depiction of the details of its execution. Like in GKAT, it is possible for distinct states to represent equivalent programs. The

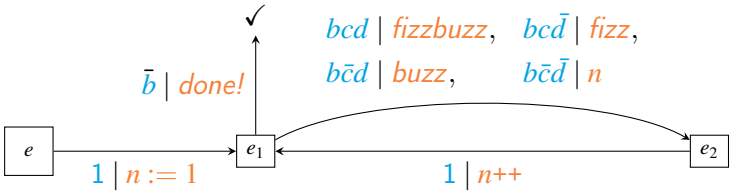



**Figure 3.7:** The automaton representing fizzbuzz2.

remaining two semantics of skip-free GKAT programs capture two ways in which skip-free programs can be equivalent: modulo bisimilarity, and modulo guarded language equivalence. Like in GKAT, the key difference between these two semantics is their ability to distinguish programs that crash early in the execution versus programs that crash later (this is apparent in the axiomatizations of both semantics). We start by presenting the language semantics, as this is the more traditional one associated with GKAT (and regular) expressions.

**Language semantics**

Formally, a *skip-free guarded trace* is a nonempty string of the form $\alpha_1 p_1 \cdots \alpha_n p_n$, where each $\alpha_i \in At$ and $p_i \in \Sigma$. Note that the only difference between guarded traces and skip-free guarded traces is the atomic test that appears at the end of a guarded trace is left implicit in a skip-free guarded trace. A *skip-free guarded language* is a set of skip-free guarded traces. Skip-free guarded languages should be thought of as sets of strings denoting successfully terminating computations.

**Definition 3.2.9** (Language acceptance)**.** In a skip-free automaton $(X, \delta_X)$ with a state $x \in X$, the *language accepted by* $x$ is the skip-free guarded language

$$\mathcal{L}(x, (X, \delta_X)) = \{\alpha_1 p_1 \cdots \alpha_n p_n \mid x \xrightarrow{\alpha_1|p_1} x_1 \to \cdots \to x_n \xrightarrow{\alpha_n|p_n} \checkmark\}$$

If $(X, \delta_X)$ is clear from context, we simply write $\mathcal{L}(x)$ instead of $\mathcal{L}(x, (X, \delta_X))$. If $\mathcal{L}(x) = \mathcal{L}(y)$, we say that $x$ and $y$ are *language equivalent*.

Each skip-free GKAT expression is a state in the automaton of expressions (Definition 3.2.5), and therefore accepts a language. As in Definition 3.2.9, the language accepted by a skip-free GKAT expression is the set of successful runs of the program it denotes. Analogously to GKAT, we can describe this language inductively.

**Lemma 3.2.10.** *Given an expression $e \in GExp_{\mathsf{sf}}$, the language accepted by $e$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$, i.e., $\mathcal{L}(e) = \mathcal{L}(e, (GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}}))$, can be computed inductively as follows:*

$$\mathcal{L}(0) = \emptyset \quad \mathcal{L}(p) = \{\alpha p \mid \alpha \in At\} \quad \mathcal{L}(e_1 +_b e_2) = b\mathcal{L}(e_1) \cup \bar{b}\mathcal{L}(e_2)$$
$$\mathcal{L}(e_1 \cdot e_2) = \mathcal{L}(e_1) \cdot \mathcal{L}(e_2) \quad \mathcal{L}({e_1}^{(b)} e_2) = \bigcup_{n \in \mathbb{N}} (b\mathcal{L}(e_1))^n \cdot \bar{b}\mathcal{L}(e_2)$$

*Here, we write $bL = \{\alpha p w \in L \mid \alpha \leq b\}$ and $L_1 \cdot L_2 = \{wu \mid w \in L_1, u \in L_2\}$, while $L^0 = \{\varepsilon\}$ (where $\varepsilon$ denotes the empty word) and $L^{n+1} = L \cdot L^n$.*

*Proof.* Since $\langle 0 \rangle$ consists of a single state with no outgoing transitions, clearly $\mathcal{L}(0) = \emptyset$. Similarly, the only outgoing transitions of each $p \in \Sigma$ are $p \xrightarrow{\alpha|p} \checkmark$, so $\mathcal{L}(p) = \{\alpha p \mid \alpha \in At\}$. Every successfully terminating path out of $e_1 +_b e_2$ is of the form $e_1 +_b e_2 \xrightarrow{\alpha|p} g_1 \xrightarrow{\alpha_1|p_1} \cdots \to g_n \xrightarrow{\alpha_n|p_n} \checkmark$ where either $\alpha \leq b$ and $e_1 \xrightarrow{\alpha|p} g_1 \xrightarrow{\alpha_1|p_1} \cdots \to g_n \xrightarrow{\alpha_n|p_n} \checkmark$ or $\alpha \leq \bar{b}$ and $e_2 \xrightarrow{\alpha|p} g_1 \xrightarrow{\alpha_1|p_1} \cdots \to g_n \xrightarrow{\alpha_n|p_n} \checkmark$. Every successfully terminating path out of $e_1 e_2$ is of the form $e_1 e_2 \xrightarrow{\alpha_1|p_1} g_1 e_2 \to \cdots \xrightarrow{\alpha_n|p_n} e_2 \xrightarrow{\alpha'_1|q_1} \cdots \xrightarrow{\alpha'_m|q_m} \checkmark$ where $e_1 \xrightarrow{\alpha_1|p_1} g_1 \xrightarrow{\alpha_1|p_1} \cdots \to g_n \xrightarrow{\alpha_n|p_n} \checkmark$ and $e_2 \xrightarrow{\alpha'_1|q_1} h_1 \xrightarrow{\alpha'_2|q_2} \cdots \to h_n \xrightarrow{\alpha'_m|q_m} \checkmark$.

Lastly, we consider ${e_1}^{(b)} e_2$. Call a cycle in an $H$-coalgebra *minimal* if every state appears at most once in the cycle. Note that every cycle is a composition of minimal cycles. Minimal cycles containing ${e_1}^{(b)} e_2$ are of the form ${e_1}^{(b)} e_2 \xrightarrow{\alpha p} g_1({e_1}^{(b)} e_2) \xrightarrow{\alpha_1|p_1} \cdots \xrightarrow{\alpha_n|p_n} {e_1}^{(b)} e_2$ where $\alpha \leq b$ and $e_1 \xrightarrow{\alpha|p} g_1 \xrightarrow{\alpha_1|p_1} \cdots \to g_n \xrightarrow{\alpha_n|p_n} \checkmark$. Successfully terminating paths from ${e_1}^{(b)} e_2$ that do not contain cycles are of the form ${e_1}^{(b)} e_2 \xrightarrow{\alpha'|q} h_1 \xrightarrow{\alpha'_1|q_1} \cdots \to h_n \xrightarrow{\alpha'_m|q_m} \checkmark$ where $e_2 \xrightarrow{\alpha'|q} h_1 \xrightarrow{\alpha'_1|q_1} \cdots \to h_n \xrightarrow{\alpha'_m|q_m} \checkmark$. Putting these together, a successfully terminating path from ${e_1}^{(b)} e_2$ is a composition of minimal cycles followed by a successfully terminating path coming from $e_2$. It follows that the guarded traces accepted by ${e_1}^{(b)} e_2$ are those of the form $w_1 \cdots w_n u$, where each $w_i \in \mathcal{L}(e_1)$ and starts with an atomic test below $b$ and $u \in \mathcal{L}(e_2)$ and starts with an atomic test below $\bar{b}$. In symbols, $\mathcal{L}({e_1}^{(b)} e_2) = \bigcup_{n \in \mathbb{N}} (b\mathcal{L}(e_1))^n \cdot \bar{b}\mathcal{L}(e_2)$. □

Lemma 3.2.10 provides a way of computing the language of an expression $e$ without having to generate the automaton for $e$.

**Bisimulation semantics**

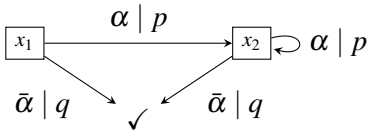
Another, finer, notion of equivalence that we can associate with skip-free automata is bisimilarity. Bisimilarity of $H$-coalgebras can be concretely characterized as follows.

**Lemma 3.2.11.** *Given skip-free automata $(X, \delta_X)$ and $(Y, \delta_Y)$, a relation $R \subseteq X \times Y$ is a bisimulation if and only if for any $(x,y) \in R$, $\alpha \in At$ and $p \in \Sigma$,*

1. *$x \downarrow \alpha$ if and only if $y \downarrow \alpha$,*
2. *$x \xrightarrow{\alpha|p} \checkmark$ if and only if $y \xrightarrow{\alpha|p} \checkmark$, and*
3. *for any $(x',y') \in R$, $x \xrightarrow{\alpha|p} x'$ if and only if $y \xrightarrow{\alpha|p} y'$.*

We call $x$ and $y$ *bisimilar* if $(x,y) \in R$ for some bisimulation $R$ and write $x \leftrightarroweq y$. The bisimilarity equivalence class of a state is called its *behaviour*.

*Example* 3.2.12. In the automaton below, $x_1$ and $x_2$ are bisimilar. This is witnessed by the bisimulation $\{(x_1,x_2),(x_2,x_2)\}$.




We can use bisimulations to witness language equivalence.

**Lemma 3.2.13.** *Let $e_1, e_2 \in GExp_{\mathsf{sf}}$. If $e_1 \leftrightarroweq e_2$, then $\mathcal{L}(e_1) = \mathcal{L}(e_2)$.*

*Proof.* Similar to the proof of Lemma 3.1.20. □

The converse of Lemma 3.2.13 is not true: Consider, for example, the program $p^{(1)}q$ that repeats the atomic action $p \in \Sigma$ indefinitely, never reaching $q$. Since

$$\mathcal{L}(p^{(1)}q) = \bigcup_{n \in \mathbb{N}} \mathcal{L}(p)^n \cdot \emptyset = \emptyset = \mathcal{L}(0)$$

we know that $\mathcal{L}(p^{(1)}q) = \mathcal{L}(0)$. But $p^{(1)}q$ and $0$ are not bisimilar, since Figure 3.6 tells us that $p^{(1)}q \xrightarrow{\alpha|p} p^{(1)}q$ whereas $0 \downarrow \alpha$.

## Axioms

Next, we give an inference system for bisimilarity and language equivalence consisting of equations and equational inference rules.

**Definition 3.2.14.** The theory sfGKAT, called *skip-free (language)* GKAT, consists of the axioms in Figure 3.8. The theory $\mathsf{sfGKAT}_0$, called *skip-free bisimulation* GKAT, is sfGKAT without the axiom (R0). Given $e, f \in GExp_{\mathsf{sf}}$, we write $e \equiv f$ if $\mathsf{sfGKAT} \vdash e = f$ and $e \equiv_0 f$ if $\mathsf{sfGKAT}_0 \vdash e = f$.

(G0) $e +_1 f = e$

(G1) $e +_b e = e$

(G2) $e_1 +_b e_2 = e_2 +_{\bar{b}} e_1$

(G3) $e_1 +_b (e_2 +_c e_3) = (e_1 +_b e_2) +_{b \vee c} e_3$

(G6) $0e = 0$

(G7) $e_1(e_2 e_3) = (e_1 e_2)e_3$

(G8) $(e_1 +_b e_2)e_3 = e_1 e_3 +_b e_2 e_3$

(FP1) $e_1{}^{(b)} e_2 = e_1(e_1{}^{(b)} e_2) +_b e_2$

(RSP) $\dfrac{g = e_1 g +_b e_2}{g = e_1{}^{(b)} e_2}$

(R0) $e0 = 0$

**Figure 3.8:** As a theory, skip-free language GKAT (sfGKAT) consists of (G0)-(G3), (G6)-(G8), (FP1), (R0), and (RSP). The theory skip-free bisimulation GKAT (sfGKAT$_0$) consists of (G0)-(G3), (G6)-(G8), (FP1), and (RSP). Note the omission of (G4), (G5), (FP2), and (DM), as well as the change from (RSP*) to (RSP).

The axiom (RSP) in Figure 3.8 is the axiom (RSP$^*$) from Figure 3.5 without the non-algebraic side condition $E(e) = 0$. It follows from an easy induction on terms that $\mathsf{BA} \vdash E(e) = 0$ holds for every skip-free GKAT expression $e$.

We also note the following theorems of sfGKAT and of sfGKAT$_0$.

**Proposition 3.2.15.** *Let $e_1, e_2, e_3, e_4 \in GExp_{\mathsf{sf}}$, $b, c, d \in BExp$. Then the following hold.*

(G3') $(e_1 +_b e_2) +_c e_3 \equiv_0 e_1 +_{b \wedge c} (e_2 +_c e_3)$

(G9) *If $b \leq c$, then $e_1 +_b e_2 \equiv_0 (e_1 +_c e_3) +_b e_2$.*

(G10) $(e_1 +_b e_2) +_c e_3 \equiv_0 (e_1 +_{b \wedge c} e_2) +_c e_3$

(BD) $(e_1 +_c e_2) +_b 0 \equiv_0 (e_1 +_b 0) +_c (e_2 +_b 0)$

(S6') $(e +_b 0) +_c 0 \equiv_0 e +_{b \wedge c} 0$

(G7') $(e_1{}^{(b)} e_2)e_3 \equiv_0 e_1{}^{(b)}(e_2 e_3)$

(FP2') $(e_1 +_b e_3)^{(b)}(e_4 +_b e_2) \equiv_0 e_1{}^{(b)} e_2$

*Proof.* For (G3'), we have

$$\begin{aligned}
(e_1 +_b e_2) +_c e_3 &\equiv_0 e_3 +_{\bar{c}} (e_2 +_{\bar{b}} e_1) && \text{(G2)} \\
&\equiv_0 (e_3 +_{\bar{c}} e_2) +_{\bar{b} \vee \bar{c}} e_1 && \text{(G3)} \\
&\equiv_0 (e_3 +_{\bar{c}} e_2) +_{\overline{(b \wedge c)}} e_1 && \text{(BA)} \\
&\equiv_0 e_1 +_{b \wedge c} (e_2 +_c e_3) && \text{(G2)}
\end{aligned}$$

For (G9), first observe that

$$\begin{aligned} e_1 +_b e_2 &\equiv_0 (e_2 +_1 g) +_{\bar{b}} e_1 && \text{(G2, G0)} \\ &\equiv_0 e_2 +_{\bar{b}} (g +_{\bar{b}} e_1) && \text{(G3',BA)} \\ &\equiv_0 (e_1 +_b g) +_b e_2 && \text{(G2)} \end{aligned}$$

Instantiating $e_1 +_b e_2 \equiv_0 (e_1 +_b g) +_b e_2$ with $g = e_1 +_c e_3$, we have

$$\begin{aligned} e_1 +_b e_2 &\equiv_0 (e_1 +_b (e_1 +_c e_3)) +_b e_2 \\ &\equiv_0 ((e_1 +_b e_1) +_c e_3) +_b e_2 && \text{(G3, BA)} \\ &\equiv_0 (e_1 +_c e_3) +_b e_2 && \text{(G1)} \end{aligned}$$

For (G10),

$$(e_1 +_b e_2) +_c e_3 \overset{\text{(G3')}}{\equiv_0} e_1 +_{b\wedge c} (e_2 +_c e_3) \overset{\text{(G3, BA)}}{\equiv_0} (e_1 +_{b\wedge c} e_2) +_c e_3$$

For (BD),

$$\begin{aligned} (e_1 +_b 0) +_c (e_2 +_b 0) &\equiv e_1 +_{b\wedge c} (0 +_c (e_2 +_b 0)) && \text{(G3')} \\ &\equiv e_1 +_{b\wedge c} ((e_2 +_b 0) +_{\bar{c}} 0) && \text{(G2)} \\ &\equiv e_1 +_{b\wedge c} (e_2 +_{b\wedge\bar{c}} (0 +_{\bar{c}} 0)) && \text{(G3')} \\ &\equiv e_1 +_{b\wedge c} (e_2 +_{b\wedge\bar{c}} 0) && \text{(G1)} \\ &\equiv (e_1 +_{b\wedge c} e_2) +_{(b\wedge c)\vee(b\wedge\bar{c})} 0 && \text{(G3)} \\ &\equiv (e_1 +_{b\wedge c} e_2) +_b 0 && \text{(BA)} \\ &\equiv (e_1 +_c e_2) +_b 0 && \text{(G10)} \end{aligned}$$

For (S6'), analogous to (S6) in Figure 3.4,

$$(e +_b 0) +_c 0 \overset{\text{(G10)}}{=} e +_{b\wedge c} (0 +_c 0) \overset{\text{(G1)}}{=} e +_{b\wedge c} 0$$

For (G7'),

$$(e_1 {}^{(b)} e_2) e_3 \equiv_0 (e_1 (e_1 {}^{(b)} e_2) +_b e_2) e_3 \qquad \text{(FP1)}$$

$$\equiv_0 e_1((e_1{}^{(b)}e_2)e_3) +_b e_2e_3 \qquad \text{(G8,G7)}$$
$$\equiv_0 e_1{}^{(b)}(e_2e_3) \qquad \text{(RSP)}$$

For (FP2'), let $g = (e_1 +_b e_3)^{(b)}(e_4 +_b e_2)$. Then

$$\begin{aligned} g &\equiv_0 (e_1 +_b e_2)g +_b (e_4 +_b e_2) && \text{(FP1)}\\ &\equiv_0 (e_1g +_b e_2g) +_b (e_4 +_b e_2) && \text{(G8)}\\ &\equiv_0 e_1g +_b e_2 && \text{(G9, with } c = b\text{, G2, BA)}\\ &\equiv_0 e_1{}^{(b)}e_2 && \text{(RSP)} \end{aligned}$$

□

*Remark* 3.2.16. The theory sfGKAT is a restriction of GKAT to $GExp_{\mathsf{sf}}$, so the derivations above are also valid in $\mathsf{GKAT}_0$. This means the identities in Proposition 3.2.15 hold for general GKAT expressions $e, f \in GExp$.

Note that the identities (BD) and (S6') of Proposition 3.2.15 can be rephrased using slightly different notation. If we write $be$ for the skip-free expression $e +_b 0$, then (BD) becomes $b(e_1 +_c e_2) \equiv_0 be_1 +_b be_2$ (*Boolean distributivity* on the left), and (S6') becomes $bce = (b \wedge c)e$.

Like in GKAT, the axiom (R0) is sound with respect to the language semantics in Definition 3.2.9 but not sound with respect to the bisimulation semantics.

**Theorem 3.2.17** (Soundness)**.** *For any* $e_1, e_2 \in GExp_{\mathsf{sf}}$,

1. *If* $e_1 \equiv_0 e_2$*, then* $e_1 \leftrightarrows e_2$.
2. *If* $e_1 \equiv e_2$*, then* $\mathcal{L}(e_1) = \mathcal{L}(e_2)$.

*Proof.* We begin with the proof that $\equiv_0$ is a bisimulation on $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. Let $e_1 \equiv_0 e_2$. We specifically that show $(e_1, e_2)$ satisfies *1.-3.* of Lemma 3.2.11 by induction on the proof of $\mathsf{sfGKAT}_0 \vdash e_1 = e_2$. In the base case, we consider the equational axioms.

- If $(e_1, e_2)$ is $(e, e +_1 f)$, $(e, e +_b e)$, $(e +_b f, f +_{\bar{b}} e)$, $(e +_b (f +_c g), (e +_b f) +_{b \vee c} g)$, $(0, 0e)$, $((e +_b f)g, eg +_b fg)$, or $(e^{(b)}f, e(e^{(b)}f) +_b f)$, then $\delta_{\mathsf{sf}}(e_1) = \delta_{\mathsf{sf}}(e_2)$. For

example, for any $\alpha \in At$,

$$\delta_{\mathsf{sf}}(e +_1 f)(\alpha) = \begin{cases} \delta_{\mathsf{sf}}(e)(\alpha) & \alpha \leq 1 \\ \delta_{\mathsf{sf}}(f)(\alpha) & \alpha \leq 0 \end{cases} = \delta_{\mathsf{sf}}(e)(\alpha)$$

because $\alpha \leq 1$ always holds. The most exciting cases are the last two. Below, missing case distinctions are assumed to be reject transitions.

$$\delta_{\mathsf{sf}}((e +_b f)g)(\alpha) = \begin{cases} (p, e'g) & e +_b f \xrightarrow{\alpha|p} e' \\ (p, g) & e +_b f \xrightarrow{\alpha|p} \checkmark \end{cases} \quad \text{(def. } \delta_{\mathsf{sf}}(- \cdot -)\text{)}$$

$$= \begin{cases} (p, e'g) & e \xrightarrow{\alpha|p} e' \text{ and } \alpha \leq b \\ (p, f'g) & f \xrightarrow{\alpha|p} f' \text{ and } \alpha \leq \bar{b} \\ (p, g) & e \xrightarrow{\alpha|p} \checkmark \text{ and } \alpha \leq b \\ (p, g) & f \xrightarrow{\alpha|p} \checkmark \text{ and } \alpha \leq \bar{b} \end{cases} \quad \text{(def. } \delta_{\mathsf{sf}}(- +_b -)\text{)}$$

$$= \begin{cases} (p, e'g) & e \xrightarrow{\alpha|p} e' \text{ and } \alpha \leq b \\ (p, g) & e \xrightarrow{\alpha|p} \checkmark \text{ and } \alpha \leq b \\ (p, f'g) & f \xrightarrow{\alpha|p} f' \text{ and } \alpha \leq \bar{b} \\ (p, g) & f \xrightarrow{\alpha|p} \checkmark \text{ and } \alpha \leq \bar{b} \end{cases} \quad \text{(rearranging)}$$

$$= \begin{cases} \delta_{\mathsf{sf}}(eg)(\alpha) & \alpha \leq b \\ \delta_{\mathsf{sf}}(fg)(\alpha) & \alpha \leq \bar{b} \end{cases} \quad \text{(def. } \delta_{\mathsf{sf}}(- \cdot -)\text{)}$$

$$= \delta_{\mathsf{sf}}(eg +_b fg)(\alpha) \quad \text{(def. } \delta_{\mathsf{sf}}(- +_b -)\text{)}$$

$$\delta_{\mathsf{sf}}(e(e^{(b)}f) +_b f)(\alpha) = \begin{cases} \delta_{\mathsf{sf}}(e(e^{(b)}f))(\alpha) & \alpha \leq b \\ \delta_{\mathsf{sf}}(f)(\alpha) & \alpha \leq \bar{b} \end{cases} \quad \text{(def. } \delta_{\mathsf{sf}}(- +_b -)\text{)}$$

$$= \begin{cases} (p, e'(e^{(b)}f))(\alpha) & e \xrightarrow{\alpha|p} e' \text{ and } \alpha \leq b \\ (p, e^{(b)}f)(\alpha) & e \xrightarrow{\alpha|p} \checkmark \text{ and } \alpha \leq b \\ \delta_{\mathsf{sf}}(f)(\alpha) & \alpha \leq \bar{b} \end{cases}$$

$$\text{(def. } \delta_{\mathsf{sf}}(- \cdot -)\text{)}$$

$$= \delta_{\mathsf{sf}}(e^{(b)}f)(\alpha) \quad \text{(def. } \delta_{\mathsf{sf}}(-^{(b)}-)\text{)}$$

- In the $(e(fg),(ef)g)$ case, let $\alpha \in At$. There are a few subcases to consider.
  - Since $e \downarrow \alpha$ if and only if $ef \downarrow \alpha$, $e(fg) \downarrow \alpha$ if and only if $(ef)g \downarrow \alpha$. That is, *1.* in Lemma 3.2.11 is satisfied by this $\alpha$.
  - If $e \xrightarrow{\alpha|p} \checkmark$, then $ef \xrightarrow{\alpha|p} f$. Therefore, $e(fg) \xrightarrow{\alpha|p} fg$ and $(ef)g \xrightarrow{\alpha|p} fg$.
  - If $e \xrightarrow{\alpha|p} e'$, then $e(fg) \xrightarrow{\alpha|p} e'(fg)$. We also know that $ef \xrightarrow{\alpha|p} e'f$, so $(ef)g \xrightarrow{\alpha|p} (e'f)g$. Since $e'(fg) \equiv_0 (e'f)g$, in conjunction with the previous subcase we see that *3.* of Lemma 3.2.11 is satisfied for this $\alpha$.

In the induction step, we consider (Con) and (RSP). The remaining rules are routine.

- We are going to show that the congruence rules preserve the properties in Lemma 3.2.11. Suppose $e_1 \equiv_0 e_2$ and $(e_1,e_2)$ satisfies the conditions of Lemma 3.2.11.
  - It suffices to consider $\alpha \leq b$ in the $(e_1 +_b g, e_2 +_b g)$ subcase. This follows directly from the induction hypothesis applied to $(e_1,e_2)$ if $\alpha \leq b$.
  - In the $(e_1g,e_2g)$ subcase, $e_1 \downarrow \alpha$ if and only if $e_2 \downarrow \alpha$ by the induction hypothesis, and $e_ig \downarrow \alpha$ if and only if $e_i \downarrow \alpha$. Similarly, $e_1 \xrightarrow{\alpha|p} \checkmark$ if and only if $e_2 \xrightarrow{\alpha|p} \checkmark$, so $e_1g \xrightarrow{\alpha|p} g$ if and only if $e_2g \xrightarrow{\alpha|p} g$. If $e_1' \equiv_0 e_2'$, then $e_1 \xrightarrow{\alpha|p} e_1'$ if and only if $e_2 \xrightarrow{\alpha|p} e_2'$. It follows that $e_1g \xrightarrow{\alpha|p} e_1'g$ if and only if $e_2g \xrightarrow{\alpha|p} e_2'g$.
  - The subcase $(ge_1,ge_2)$ is easily seen from the $(g,g)$ base case.
  - Consider $({e_1}^{(b)}g,{e_2}^{(b)}g)$. We only cover the $\alpha \leq b$ case here because ${e_1}^{(b)}g$ and ${e_2}^{(b)}g$ have the same outgoing $\alpha \leq \bar{b}$ transitions. The condition *1.* of Lemma 3.2.11 follows from the fact that ${e_i}^{(b)}g \downarrow \alpha$ if and only if $e_i \downarrow \alpha$. Similarly, ${e_i}^{(b)}g \xrightarrow{\alpha|p} {e_i}^{(b)}g$ if and only if $e_i \xrightarrow{\alpha|p} \checkmark$ for $i = 1,2$. Finally, if $e_i \xrightarrow{\alpha|p} e_i'$ for $i = 1,2$ and $e_1' \equiv_0 e_2'$, then $e_1'({e_1}^{(b)}g) \equiv_0 e_2'({e_2}^{(b)}g)$ and ${e_i}^{(b)}g \xrightarrow{\alpha|p} e_i'({e_i}^{(b)}g)$ for $i = 1,2$.
- Let $g \equiv_0 eg +_b f$ and assume for an induction hypothesis that $(g, eg +_b f)$ satisfies *1.-3.* in Lemma 3.2.11. Then $g \downarrow \alpha$ if and only if either $\alpha \leq b$ and $e \downarrow \alpha$ or $\alpha \leq \bar{b}$ and $f \downarrow \alpha$. Since the latter crashes are the same as those of $e^{(b)}f \downarrow \alpha$, we have verified *1.* in Lemma 3.2.11.

  We also know that $g \xrightarrow{\alpha|p} \checkmark$ if and only if $\alpha \leq \bar{b}$ and $f \xrightarrow{\alpha|p} \checkmark$. The latter successful termination conditions are those of $e^{(b)}f \xrightarrow{\alpha|p} \checkmark$.

Lastly, by the induction hypothesis, $g \xrightarrow{\alpha|p} g'$ if and only if one of the following occurs:

- $\alpha \leq b$ and $e \xrightarrow{\alpha|p} e'$. This is equivalent to $e^{(b)}f \xrightarrow{\alpha|p} e'(e^{(b)}f)$. On the other hand, $e'g \equiv_0 e'(e^{(b)}f)$, and by the induction hypothesis $g' \equiv_0 e'g$.
- $\alpha \leq b$ and $e \xrightarrow{\alpha|p} \checkmark$. This means that $e^{(b)}f \xrightarrow{\alpha|p} e^{(b)}f$. By the induction hypothesis, $g' \equiv_0 e^{(b)}f \equiv_0 g$.
- $\alpha \leq \bar{b}$ and $f \xrightarrow{\alpha|p} f'$. In this case, by the induction hypothesis, $g' \equiv_0 f'$.

We have shown that $g \xrightarrow{\alpha|p} g'$ and $e^{(b)}f \xrightarrow{\alpha|p} e'$ implies $g' \equiv_0 e'$. This establishes *3.* in Lemma 3.2.11.

Now we verify that $\equiv$ is sound with respect to language equivalence. Again, we show that $e_1 \equiv e_2$ implies $\mathcal{L}(e_1) = \mathcal{L}(e_2)$ by induction on the proof of $e_1 = e_2$ from the rules in Figure 3.5. We have already seen that if $e_1 \equiv_0 e_2$ implies $e_1 \leftrightarrows e_2$, so by Lemma 3.2.13 we know that $e_1 \equiv_0 e_2$ implies $\mathcal{L}(e_1) = \mathcal{L}(e_2)$. This handles most of the base case. Reflexivity, symmetry, and transitivity are handled by the fact that language equivalence is the kernel of $\mathcal{L}$. This leaves us with $\mathcal{L}(e0) = \mathcal{L}(0)$, which can be seen from Lemma 3.2.10:

$$\mathcal{L}(e0) = \mathcal{L}(e) \cdot \mathcal{L}(0) = \mathcal{L}(e) \cdot \emptyset = \emptyset = \mathcal{L}(0)$$

In the inductive step, we need to consider the transitivity, congruence, and Horn rules. Congruence is a consequence of Lemma 3.2.10. Transitivity follows from the fact that language equivalence is the kernel of $\mathcal{L}$. Soundness of the rule $g = eg +_b f \implies g = e^{(b)}f$ follows from Lemma 3.2.10 and the fact that for any languages $U, L, W \subseteq (At \cdot \Sigma)^+$ (i.e., not containing the empty word),

$$L = UL \cup W \implies L = U^*W \qquad (\heartsuit)$$

Now, if $\mathcal{L}(g) = \mathcal{L}(eg +_b f)$, then because $\mathcal{L}(eg +_b f) = b\mathcal{L}(e) \cdot \mathcal{L}(g) \cup \bar{b}\mathcal{L}(f)$, $\mathcal{L}(g) = (b\mathcal{L}(e))^* \cdot \bar{b}\mathcal{L}(f) = \mathcal{L}(e^{(b)}f)$ by $(\heartsuit)$. □

**The main results of the chapter**

I consider the next two results, which are jointly converse to Theorem 3.2.17, to be the main theorems of this chapter. They state that the axioms in Figure 3.5 are

*complete* for bisimilarity and language equivalence respectively—that they describe a complete set of program transformations for skip-free GKAT.

**Theorem 3.2.18** (Completeness I)**.** *Let* $e_1, e_2 \in GExp_{\mathsf{sf}}$*. If* $e_1 \leftrightarrows e_2$*, then* $e_1 \equiv_0 e_2$*.*

**Theorem 3.2.19** (Completeness II)**.** *Let* $e_1, e_2 \in GExp_{\mathsf{sf}}$*. If* $\mathcal{L}(e_1) = \mathcal{L}(e_2)$*, then* $e_1 \equiv e_2$*.*

We prove Theorem 3.2.18 in Section 3.3 by drawing a formal analogy between skip-free GKAT and the one-free regular expressions modulo bisimilarity from Chapter 2. We delay the proof of Theorem 3.2.19 to Section 3.4, which uses a technique similar to the pruning method of Section 3.1.6.

## 3.3 Completeness for Skip-free Bisimulation GKAT

This section is dedicated to the proof of our first completeness result, Theorem 3.2.18, which says that the axioms of Figure 3.5 (excluding (R0)) are complete with respect to bisimilarity in skip-free GKAT. Our proof strategy is a reduction to the completeness theorem for one-free regular expressions modulo bisimilarity (Theorem 2.4.13).

The key objects of interest in the reduction are a pair of translations: One translation turns skip-free GKAT expressions into one-free regular expressions and maintains bisimilarity, and the other translation turns (certain) one-free regular expressions into skip-free GKAT expressions and maintains provable equivalence.

We begin with a transformation between automata and labelled transition systems that preserves and reflects bisimilarity. We then introduce the syntactic translations and present the completeness proof.

### 3.3.1 From skip-free automata to labelled transition systems

Recall that a prechart is an $L$-coalgebra, where $LX = \mathcal{P}_\omega(Act \times (\checkmark + X))$ and $Act$ is a fixed set of action symbols. In the reduction from skip-free GKAT to one-free regular expressions modulo bisimilarity, we fix the set of action symbols $Act = At \cdot \Sigma$, pairs of atomic tests and primitive programs $\Sigma$. With this set of action symbols, we can easily transform a skip-free automaton into a prechart (Definition 2.2.4), essentially by turning each $\xrightarrow{\alpha|p}$ transition into a $\xrightarrow{\alpha p}$ transition.

**Definition 3.3.1.** Given a set $X$, we define

$$\mathsf{grph}_X : HX \to LX \qquad \mathsf{grph}_X(\theta) = \{(\alpha p, x) \mid \theta(\alpha) = (p, x)\}$$

Each $\mathsf{grph}_X$ is injective and the definition of $\mathsf{grph}_X$ is natural in $X$. As its name suggests, $\mathsf{grph}_X(\theta)$ is essentially the graph of $\theta$ when viewed as a partial function from $At$ to $\Sigma \times (\checkmark + X)$. By Lemma 2.4.8, this implies that the transformation $\mathsf{grph}_*$ of skip-free automata into precharts preserves bisimilarity.

As in Lemma 2.4.8, the natural transformation $\mathsf{grph}$ defines a functor. Given a skip-free automaton $(X, \delta_X)$ and a homomorphism $h\colon (X,\delta_X) \to (Y,\delta_Y)$ between skip-free automata, we define

$$\mathsf{grph}_*\colon \mathrm{Coalg}(H) \to \mathrm{Coalg}(L)$$
$$\mathsf{grph}_*(X,\delta_X) = (X, \mathsf{grph}_X \circ \delta_X) \quad \mathsf{grph}_*(h) = h$$

where $L$ is the coalgebraic signature of precharts.

Leading up to the proof of Theorem 3.2.18, we also need that $\mathsf{grph}_*$ *reflects* bisimilarity. In particular, we are going to undo the effect of $\mathsf{grph}_*$ on skip-free automata with a transformation that takes every prechart of the form $\mathsf{grph}_*(X,\delta_X)$ to its underlying skip-free automaton $(X,\delta_X)$.

The precharts that can be written in the form $\mathsf{grph}_*(X,\delta_X)$ for some skip-free automaton $(X,\delta_X)$ are precisely those that are deterministic in a particular sense. Call a set $U \in \mathcal{P}(At \cdot \Sigma \times (\checkmark + X))$ *graph-like* if whenever $(\alpha p, x) \in U$ and $(\alpha q, y) \in U$, then $p = q$ and $x = y$. Graph-like sets are in one-to-one correspondence with the graphs of partial functions of the form $At \rightharpoonup \Sigma \times (\checkmark + X)$. A prechart $(S,\tau)$ is *deterministic* if $\tau(x)$ is graph-like for every $x \in S$.

**Lemma 3.3.2.** *A prechart $(S,\tau)$ is deterministic if and only if $(S,\tau) = \mathsf{grph}_*(X,\delta_X)$ for some skip-free automaton $(X,\delta_X)$.*

*Proof.* For any set $X$, let $L_{\mathrm{det}}X = \{U \in \mathcal{P}(At \cdot \Sigma \times (\checkmark + X)) \mid U \text{ is graph-like}\}$ and define $\mathsf{func}_X\colon L_{\mathrm{det}}X \to (\bot + \Sigma \times (\checkmark + X))^{At}$ by

$$\mathsf{func}_X(U)(\alpha) = \begin{cases} (p,x) & (\alpha p, x) \in U \\ \bot & \text{otherwise} \end{cases} \tag{3.12}$$

This defines a natural transformation $\mathsf{func}\colon L_{\mathrm{det}} \Rightarrow H$.

Now let $(S,\tau)$ be a deterministic prechart. Then $\mathsf{func}_*(S,\tau) = (S, \mathsf{func}_S \circ \tau)$ is a skip-free automaton such that $\mathsf{grph}_*\mathsf{func}_*(S,\tau) = (S,\tau)$, because $\mathsf{grph}_S \circ \mathsf{func}_S = \mathrm{id}_{L_{\mathrm{det}}(S)}$.

Conversely, if $(S,\tau) = \mathsf{grph}_*(S,\delta_S)$ and $x \in S$, then $\tau(x) = \mathsf{grph}_S \circ \delta_S(x)$. This implies $\tau(x)$ is graph-like, so $(S,\tau)$ is deterministic. □

Using $\mathsf{grph}_*$ and $\mathsf{func}_*$, (3.12), we can prove the following.

**Lemma 3.3.3.** *Let $x,y \in X$, and $(X,\delta_X)$ be a skip-free automaton. Then $x \leftrightarrows y$ in $(X,\delta_X)$ if and only if $x \leftrightarrows y$ in $\mathsf{grph}_*(X,\delta_X)$.*

*Proof.* The forward direction is Lemma 2.4.8 item 2. For the converse, we first observe that if $(R,\delta_R)$ is a bisimulation on $\mathsf{grph}_*(X,\delta_X)$, then $(R,\delta_R)$ is deterministic. Indeed, if $(\alpha p,(x',y')),(\alpha q,(x'',y'')) \in \delta_R(x,y)$, then $(\alpha p,x'),(\alpha q,x'') \in \delta_X(x)$. Since $\mathsf{grph}_*(X,\delta_X)$ is deterministic, $p = q$ and $x' = x''$. Similarly, $y' = y''$. Applying $\mathsf{func}_*$ to $(R,\delta_R)$, we obtain a bisimulation $(R,\mathsf{func}_R \circ \delta_R)$ on $(X,\delta_X)$ containing the same pairs. It follows that if $x \leftrightarrows y$ in $\mathsf{grph}_*(X,\delta_X)$, then $x \leftrightarrows y$ in $(X,\delta_X)$. □

### 3.3.2 Translating Syntax

After fixing $Act = At \cdot \Sigma$, the set $StExp$ of one-free regular expressions is given by

$$StExp \ni r_1, r_2 ::= 0 \mid \alpha p \in At \cdot \Sigma \mid r_1 + r_2 \mid r_1 r_2 \mid r_1 * r_2$$

Recall that we say $r_1$ and $r_2$ are *provably equivalent* and write $r_1 \equiv_* r_2$ if $\mathsf{1fMil} \vdash r_1 = r_2$, where 1fMil is the theory in Definition 2.2.14.

We can mimic the transformation of skip-free automata into deterministic labelled transition systems and vice-versa by a pair of syntactic translations going back and forth between skip-free GKAT expressions and certain one-free star expressions. Similar to how only some labelled transition systems can be turned into skip-free automata, only some one-free star expressions have corresponding skip-free GKAT expressions, what we also call the deterministic ones.

The definition of deterministic expressions requires the following notation.

**Definition 3.3.4.** Given a test $b \in BExp$, we define $b \cdot r$ inductively on $r \in StExp$ by

$$b \cdot 0 = 0 \qquad b \cdot (r_1 + r_2) = b \cdot r_1 + b \cdot r_2$$

$$b \cdot \alpha p = \begin{cases} \alpha p & \alpha \leq b \\ 0 & \alpha \leq \bar{b} \end{cases} \qquad \begin{aligned} b \cdot (r_1 r_2) &= (b \cdot r_1) r_2 \\ b \cdot (r_1 * r_2) &= (b \cdot r_1)(r_1 * r_2) + b \cdot r_2 \end{aligned}$$

for any $\alpha p \in At \cdot \Sigma$ and $r_1, r_2 \in StExp$.

**Definition 3.3.5.** The set of *deterministic* one-free star expressions is the smallest subset $Det \subseteq StExp$ such that

- $0 \in Det$,
- $\alpha p \in Det$ for any $\alpha \in At$ and $p \in \Sigma$,
- if $r_1, r_2 \in Det$, then $r_1 r_2 \in Det$, and
- if $r_1, r_2 \in Det$, $b \in BExp$, and $r_1 \equiv_* b \cdot r_1$ and $r_2 \equiv_* \bar{b} \cdot r_2$, then $r_1 + r_2, r_1 * r_2 \in Det$.

In particular, if $r_1, r_2 \in Det$ and $b \in BExp$, then $b \cdot r_1 + \bar{b} \cdot r_2, (b \cdot r_1) * (\bar{b} \cdot r_2) \in Det$.

**From $GExp_{\mathsf{sf}}$ to $Det$**

We now present the translations of skip-free expressions to deterministic one-free star expressions.

**Definition 3.3.6.** We define the *translation* function $gtr\colon GExp_{\mathsf{sf}} \to Det$ by

$$gtr(0) = 0 \qquad gtr(p) = \sum_{\alpha \in At} \alpha p \qquad gtr(e_1 +_b e_2) = b \cdot gtr(e_1) + \bar{b} \cdot gtr(e_2)$$

$$gtr(e_1 \cdot e_2) = gtr(e_1) gtr(e_2) \qquad gtr(e_1 {}^{(b)} e_2) = (b \cdot e_1) * (\bar{b} \cdot e_2)$$

for any $b \in BExp$, $p \in \Sigma$, $e_1, e_2 \in GExp$.

*Remark* 3.3.7. In Definition 3.3.6, we make use of a generalized sum $\sum_{\alpha \in At}$. Technically, this requires we fix an enumeration of $At$ ahead of time, say $At = \{\alpha_1, \ldots, \alpha_n\}$, at which point we can define $\sum_{\alpha \in At} r_a = r_{\alpha_1} + \cdots + r_{\alpha_n}$. Of course, $+$ is commutative and associative up to provable equivalence of regular expressions $\equiv_*$ (see Section 2.1), so the actual ordering of this sum does not matter as far as equivalence is concerned.

An analogue of this generalized sum is also used for skip-free GKAT expressions, which we inductively define by

$$\sum_{\alpha \leq 0} \alpha \cdot e_\alpha = 0 \qquad \sum_{\alpha \leq b} \alpha \cdot e_\alpha = e_\beta +_\beta \sum_{\alpha \leq b \wedge \bar{\beta}} \alpha \cdot e_\alpha$$

such that $\beta \leq b \in BExp$ for some $\beta \in At$, and given an $e_\alpha \in GExp_{\mathsf{sf}}$ for each $\alpha \in At$. This also technically requires that we enumerate $At$, but by (G2) and (G3), any two choices in enumeration are $\equiv_0$-equivalent.

The most important feature of this translation is that it preserves bisimilarity.

**Lemma 3.3.8.** *The graph of the translation function $gtr$ is a bisimulation of labelled transition systems between $\mathsf{grph}_*(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ and $(StExp, \ell)$. Consequently, if $e_1 \leftrightarrows e_2$ in $\mathsf{grph}_*(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$, then $gtr(e_1) \leftrightarrows gtr(e_2)$ in $(StExp, \ell)$.*

*Proof.* By Lemma 2.2.12, it suffices to show that $gtr\colon \mathsf{grph}_*(GExp_{\mathsf{sf}}, \delta) \to (StExp, \ell)$ is an $L$-coalgebra homomorphism. We show by induction on $e \in GExp_{\mathsf{sf}}$ that

$$\ell(gtr(e)) = L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(e)$$

Note that the set on the right-hand side is computed

$$\begin{aligned} L(gtr) \circ \mathsf{grph}(\theta) = \{(\alpha p, \checkmark) \mid \theta(\alpha) = (p, \checkmark)\} \\ \cup \{(\alpha p, gtr(g)) \mid \theta(\alpha) = (p, g) \in \Sigma \times GExp_{\mathsf{sf}}\} \end{aligned} \tag{3.13}$$

for $\theta \in H(GExp_{\mathsf{sf}})$. We also use the following identity, derived inductively on $r$.

$$\ell(b \cdot r) = \{(\alpha p, \xi) \mid (\alpha p, \xi) \in \ell(r) \text{ and } \alpha \leq b\} \tag{3.14}$$

For the base case, we consider $e = 0$ and $e = p \in \Sigma$. We have $\ell(gtr(0)) = \ell(0) = \emptyset$ on the one hand, and on the other, $\delta_{\mathsf{sf}}(0) = \lambda\alpha.\bot$. Hence, $L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(0) = \emptyset$ as in (3.13). For $e = p$, we have $\ell(gtr(p)) = \{(\alpha p, \checkmark) \mid \alpha \in At\}$ on the one hand, and on the other $L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(p) = \{(\alpha p, \checkmark) \mid \alpha \in At\}$ on the other by (3.13). For the inductive step, assume $\ell(gtr(e_i)) = L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_i)$ for $i \in \{1, 2\}$. We have

$$\begin{aligned} \ell(gtr(e_1 +_b e_2)) &= \ell(b \cdot gtr(e_1) + \bar{b} \cdot gtr(e_2)) \\ &= \ell(b \cdot gtr(e_1)) \cup \ell(\bar{b} \cdot gtr(e_2)) \\ &= \{(\alpha p, \xi) \in \ell(gtr(e_1)) \mid \alpha \leq b\} \\ &\qquad \cup \{(\alpha p, \xi) \in \ell(gtr(e_2)) \mid \alpha \leq \bar{b}\} && (3.14) \\ &= \{(\alpha p, \xi) \in L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_1) \mid \alpha \leq b\} \\ &\qquad \cup \{(\alpha p, \xi) \in L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_2) \mid \alpha \leq \bar{b}\} && \text{(ind. hyp.)} \\ &= \{(\alpha p, \checkmark) \mid (\alpha p, \checkmark) \in \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_1) \text{ and } \alpha \leq b\} \\ &\qquad \cup \{(\alpha p, gtr(g)) \mid (\alpha p, g) \in \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_1) \text{ and } \alpha \leq b\} \end{aligned}$$

$$
\begin{aligned}
&\cup\{(\alpha p,\checkmark) \mid (\alpha p,\checkmark) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_2) \text{ and } \alpha \leq \bar{b}\}\\
&\cup\{(\alpha p, gtr(g)) \mid (\alpha p, g) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1) \text{ and } \alpha \leq \bar{b}\}\\
&= \{(\alpha p,\checkmark) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p,\checkmark) \text{ and } \alpha \leq b\}\\
&\quad\cup\{(\alpha p, gtr(g)) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p,g) \text{ and } \alpha \leq b\}\\
&\quad\cup\{(\alpha p,\checkmark) \mid \delta_{\mathsf{sf}}(e_2)(\alpha) = (p,\checkmark) \text{ and } \alpha \leq \bar{b}\}\\
&\quad\cup\{(\alpha p, gtr(g)) \mid \delta_{\mathsf{sf}}(e_2)(\alpha) = (p,g) \text{ and } \alpha \leq \bar{b}\}\\
&= \{(\alpha p,\checkmark) \mid \delta_{\mathsf{sf}}(e_1 +_b e_2)(\alpha) = (p,\checkmark)\}\\
&\quad\cup\{(\alpha p, gtr(g)) \mid \delta_{\mathsf{sf}}(e_1 +_b e_2)(\alpha) = (p,g)\}\\
&= L(gtr)\circ\mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1 +_b e_2) \qquad (3.13)\\
\ell(gtr(e_1e_2)) &= \ell(gtr(e_1)gtr(e_2))\\
&= \{(\alpha p, gtr(e_2)) \mid (\alpha p,\checkmark) \in \ell(gtr(e_1))\}\\
&\quad\cup\{(\alpha p, s\ gtr(e_2)) \mid (\alpha p, s) \in \ell(gtr(e_1))\}\\
&= \{(\alpha p, gtr(e_2)) \mid (\alpha p,\checkmark) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1)\}\\
&\quad\cup\{(\alpha p, gtr(g)gtr(e_2)) \mid (\alpha p, g) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1)\} \quad \text{(ind. hyp.)}\\
&= \{(\alpha p, gtr(e_2)) \mid (\alpha p,\checkmark) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1)\}\\
&\quad\cup\{(\alpha p, gtr(ge_2)) \mid (\alpha p, g) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1)\}\\
&= \{(\alpha p, gtr(e_2)) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p,\checkmark)\}\\
&\quad\cup\{(\alpha p, gtr(ge_2)) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p,g)\}\\
&= L(gtr)\circ\mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1e_2) \qquad (3.13)
\end{aligned}
$$

Finally, in the loop case, let $e = {e_1}^{(b)}e_2$ and compute

$$
\begin{aligned}
\ell(gtr(e)) &= \ell((b\cdot gtr(e_1)) * (\bar{b}\cdot gtr(e_2)))\\
&= \{(\alpha p, gtr(e)) \mid (\alpha p,\checkmark) \in \ell(gtr(e_1)) \text{ and } \alpha \leq b\}\\
&\quad\cup\{(\alpha p, s\ gtr(e)) \mid (\alpha p, s) \in \ell(gtr(e_1)) \text{ and } \alpha \leq b\}\\
&\quad\cup\{(\alpha p, s) \mid (\alpha p, s) \in \ell(gtr(e_2)) \text{ and } \alpha \leq \bar{b}\}\\
&\quad\cup\{(\alpha p,\checkmark) \mid (\alpha p,\checkmark) \in \ell(gtr(e_2)) \text{ and } \alpha \leq \bar{b}\}\\
&= \{(\alpha p, gtr(e)) \mid (\alpha p,\checkmark) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1) \text{ and } \alpha \leq b\}\\
&\quad\cup\{(\alpha p, gtr(g)\ gtr(e)) \mid (\alpha p, g) \in \mathsf{grph}\circ\delta_{\mathsf{sf}}(e_1) \text{ and } \alpha \leq b\}
\end{aligned}
$$

$$\cup \{(\alpha p, gtr(g)) \mid (\alpha p, g) \in \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_2) \text{ and } \alpha \leq \bar{b}\}$$
$$\cup \{(\alpha p, \checkmark) \mid (\alpha p, \checkmark) \in \mathsf{grph} \circ \delta_{\mathsf{sf}}(e_2) \text{ and } \alpha \leq \bar{b}\} \qquad \text{(ind. hyp.)}$$
$$= \{(\alpha p, gtr(e)) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, \checkmark) \text{ and } \alpha \leq b\}$$
$$\cup \{(\alpha p, gtr(ge)) \mid \delta_{\mathsf{sf}}(e_1)(\alpha) = (p, g) \text{ and } \alpha \leq b\}$$
$$\cup \{(\alpha p, gtr(g)) \mid \delta_{\mathsf{sf}}(e_2)(\alpha) = (p, g) \text{ and } \alpha \leq \bar{b}\}$$
$$\cup \{(\alpha p, \checkmark) \mid \delta_{\mathsf{sf}}(e_2)(\alpha) = (p, \checkmark) \text{ and } \alpha \leq \bar{b}\}$$
$$= L(gtr) \circ \mathsf{grph} \circ \delta_{\mathsf{sf}}(e) \qquad (3.13)$$

□

**From** *Det* **to** $GExp_{\mathsf{sf}}$

We also define a *back translation* function $rtg\colon Det \to GExp_{\mathsf{sf}}$ by induction on its argument. Looking at Definition 3.3.5, one might be tempted to write

$$rtg(b \cdot r_1 + \bar{b} \cdot r_2) = rtg(r_1) +_b rtg(r_2)$$

but it is possible for there to be distinct $b, c \in BExp$ such that $b \cdot r_1 + \bar{b} \cdot r_2 = c \cdot r_1 + \bar{c} \cdot r_2$ (i.e., they are the same expression). This means there is a choice to be made in the translation of $b \cdot r_1 + \bar{b} \cdot r_2$. For example, given $p, q \in \Sigma$ and $\alpha_1, \alpha_2, \alpha_3 \in At$, if $b = \alpha_1 \vee \alpha_2$ and $c = \alpha_1$, then $b \cdot \alpha_1 p + \bar{b} \cdot \alpha_3 q = c \cdot \alpha_1 p + \bar{c} \cdot \alpha_3 q$. As it turns out, there is a canonical translation of terms of the form $b \cdot r_1 + \bar{b} \cdot r_2$.

**Definition 3.3.9.** Write $r_1 \perp_b r_2$ and say that $r_1, r_2 \in StExp$ are *separated by* $b \in BExp$ if $r_1 \equiv_* b \cdot r_1$ and $r_2 \equiv_* \bar{b} \cdot r_2$. If such a $b$ exists we say that $r_1$ and $r_2$ are *separated* and write $r_1 \perp r_2$.

Another way to define *Det* is to say that *Det* is the smallest subset of *StExp* containing 0 and $At \cdot \Sigma$ that is closed under sequential composition and closed under unions and stars of separated one-free star expressions.

Suppose $r_1$ and $r_2$ are separated by both $b$ and $c$. Then one can prove that $(b \vee c) \cdot r_1 \equiv_* b \cdot r_1 + c \cdot r_1 \equiv_* r_1$ and $\overline{(b \vee c)} \cdot r_2 = (\bar{b} \wedge \bar{c}) \cdot r_2 \equiv_* \bar{b} \cdot (\bar{c} \cdot r_2) \equiv_* r_2$, so $r_1$ and $r_2$ are separated by $b \vee c$ as well. Since there are only finitely many Boolean expressions up to equivalence, there is a maximal (weakest) test $b(r_1, r_2) \in BExp$ such that $r_1$ and $r_2$ are separated by $b(r_1, r_2)$. This allows us to define a back-translation from

deterministic regular expressions to skip-free GKAT expressions.

**Definition 3.3.10.** Given separated $r_1, r_2 \in \mathit{RExp}$, write $b(r_1, r_2)$ for the maximal test separating $r_1$ from $r_2$. The *back translation* $rtg\colon \mathit{Det} \to \mathit{GExp}_{\mathsf{sf}}$ is defined by

$$rtg(0) = 0 \qquad rtg(\alpha p) = p +_\alpha 0 \qquad rtg(r_1 + r_2) = rtg(r_1) +_{b(r_1,r_2)} rtg(r_2)$$
$$rtg(r_1 r_2) = rtg(r_1) \cdot rtg(r_2) \qquad rtg(r_1 * r_2) = rtg(r_1)^{(b(r_1,r_2))} rtg(r_2)$$

for any $r_1, r_2 \in \mathit{StExp}$. In the union and star cases, $r_1$ and $r_2$ are separated by definition of $\mathit{Det}$, so that $b(r_1, r_2)$ is well-defined.

To define the back-translation $rtg$, we had to choose a separating test in the translations of $r_1 + r_2$ and $r_1 * r_2$ (we chose the maximal separating test). A consequence of the next lemma is that any other choice in separating test would have produced provably equivalent skip-free expressions.

**Lemma 3.3.11.** *Let $b$ be any test. If $r \perp_b s$, then $rtg(r+s) \equiv_0 rtg(r) +_b rtg(s)$ and $rtg(r*s) \equiv_0 rtg(r)^{(b)} rtg(s)$.*

To prove Lemma 3.3.11, we need the following intermediate property.

**Lemma 3.3.12.** *If $r \equiv_* b \cdot r$, then $rtg(r) \equiv_0 rtg(r) +_b 0$.*

*Proof.* By induction on $r \in \mathit{Det}$.

- In the first base case, $r = 0$. Of course, $b \cdot 0 = 0$ for any $b \in \mathit{BExp}$. By (G1), $rtg(r) +_b 0 = 0 +_b 0 \equiv_0 0 = rtg(0)$.
- For the second base case, consider $\alpha p$ for some $\alpha \in \mathit{At}$ and $p \in \Sigma$. If $\alpha \leq \bar{b}$, then $b \cdot \alpha p = 0 \not\equiv_* \alpha p$. Otherwise, $\alpha p \equiv_* b \cdot \alpha p$ and

$$\begin{aligned} rtg(\alpha p) +_b 0 &= (p +_\alpha 0) +_b 0 \\ &\equiv_0 p +_\alpha (0 +_b 0) && \text{(G3', BA)} \\ &\equiv_0 p +_\alpha 0 && \text{(G1)} \\ &= rtg(\alpha p) \end{aligned}$$

- For the first inductive case, suppose $r_1 \perp_d r_2$ and $b \cdot (r_1 + r_2) \equiv_* r_1 + r_2$. Then

$$(b \wedge d) \cdot (r_1 + r_2) \equiv_* b \cdot (d \cdot r_1 + d \cdot r_2) \equiv_* b \cdot (d \cdot r_1 + 0) \equiv_* b \cdot (d \cdot r_1) \equiv_* b \cdot r_1 \equiv_* r_1$$

and similarly $r_2 \equiv_* (b \wedge \bar{d}) \cdot (r_1 + r_2)$. This implies $r_i \equiv_* b \cdot r_i$ for $i \in \{1,2\}$, so

$$\begin{aligned} rtg(r_1 + r_2) +_b 0 &\equiv_0 (rtg(r_1) +_{b(r_1,r_2)} rtg(r_2)) +_b 0 \\ &\equiv_0 (rtg(r_1) +_b 0) +_{b(r_1,r_2)} (rtg(r_2) +_b 0) && \text{(BD)} \\ &\equiv_0 rtg(r_1) +_{b(r_1,r_2)} rtg(r_2) && \text{(ind. hyp.)} \\ &= rtg(r_1 + r_2) \end{aligned}$$

- For the sequential composition case, if $r_1 r_2 \equiv_* b \cdot (r_1 r_2)$, then we need to argue that $r_1 \equiv_* b \cdot r_1$. This actually follows from Theorem 2.2.18, because

$$\begin{aligned} r_1 \xrightarrow{\alpha p} \checkmark \quad &\text{iff} \quad r_1 r_2 \xrightarrow{\alpha p} r_2 && \text{(def. } \ell\text{)} \\ &\text{iff} \quad b \cdot (r_1 r_2) \xrightarrow{\alpha p} r_2 && \text{(assm., soundness)} \\ &\text{iff} \quad (b \cdot r_1) r_2 \xrightarrow{\alpha p} r_2 && \text{(def. } b \cdot -\text{)} \\ &\text{iff} \quad b \cdot r_1 \xrightarrow{\alpha p} \checkmark && \text{(def. } \ell\text{)} \\ r_1 \xrightarrow{\alpha p} s \quad &\text{iff} \quad r_1 r_2 \xrightarrow{\alpha p} s r_2 && \text{(def. } \ell\text{)} \\ &\text{iff} \quad b \cdot (r_1 r_2) \xrightarrow{\alpha p} s r_2 && \text{(assm., soundness)} \\ &\text{iff} \quad (b \cdot r_1) r_2 \xrightarrow{\alpha p} s r_2 && \text{(def. } b \cdot -\text{)} \\ &\text{iff} \quad b \cdot r_1 \xrightarrow{\alpha p} s && \text{(def. } \ell\text{)} \end{aligned}$$

These observations are behind the following derivation

$$\begin{aligned} r_1 &\equiv_* \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s && \text{(Theorem 2.2.18)} \\ &\equiv_* \sum_{b \cdot r_1 \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{b \cdot r_1 \xrightarrow{\alpha p} s} \alpha p s \\ &\equiv_* b \cdot r_1 && \text{(Theorem 2.2.18)} \end{aligned}$$

Hence, $r_1 \equiv_* b \cdot r_1$. This allows us to deduce

$$\begin{aligned} rtg(r_1 r_2) = rtg(r_1) rtg(r_2) &\equiv_0 (rtg(r_1) +_b 0) rtg(r_2) && \text{(ind. hyp.)} \\ &\equiv_0 rtg(r_1) rtg(r_2) +_b 0 && \text{(G8,G6)} \\ &= rtg(r_1 r_2) +_b 0 \end{aligned}$$

- For the star case, assume $b \cdot (r_1 * r_2) \equiv_* r_1 * r_2$ for $i = 1,2$ and that $r_1$ and $r_2$ are separated by $d$. Notice that this entails

$$r_1(r_1 * r_2) + r_2 \equiv_* r_1 * r_2 \equiv_* b \cdot (r_1 * r_2) \equiv_* b \cdot (r_1(r_1 * r_2) + r_2)$$

It follows that $b \cdot r_2 \equiv_* r_2$ and $b \cdot (r_1(r_1 * r_2)) \equiv_* r_1(r_1 * r_2)$, like in the sum case. It then follows that $b \cdot r_1 \equiv_* r_1$, by the same reasoning in the sequential composition case. The induction hypothesis tells us $rtg(r_i) \equiv_0 rtg(r_i) +_b 0$ for $i \in \{1,2\}$. So,

$$\begin{aligned}
rtg(r_1 * r_2) &\equiv_0 rtg(r_1)rtg(r_1 * r_2) +_{b(r_1,r_2)} rtg(r_2) && \text{(FP1)}\\
&\equiv_0 (rtg(r_1) +_b 0)rtg(r_1 * r_2) +_{b(r_1,r_2)} (rtg(r_2) +_b 0) && \text{(ind. hyp.)}\\
&\equiv_0 (rtg(r_1)rtg(r_1 * r_2) +_b 0) +_{b(r_1,r_2)} (rtg(r_2) +_b 0) && \text{(G8,G6)}\\
&\equiv_0 (rtg(r_1)rtg(r_1 * r_2) +_{b(r_1,r_2)} rtg(r_2)) +_b 0 && \text{(BD)}\\
&\equiv_0 rtg(r_1 * r_2) +_b 0 && \text{(FP1)}
\end{aligned}$$

□

We are now ready to prove Lemma 3.3.11.

*Proof of Lemma 3.3.11.* Let $r \perp_b s$ and set $c = b(r,s)$, the maximal test separating $r$ from $s$. The key insight here is that $b \leq c$. With this observation in hand,

$$\begin{aligned}
rtg(r+s) &= rtg(r) +_c rtg(s)\\
&\equiv_0 (rtg(r) +_b 0) +_c rtg(s) && \text{(Lemma 3.3.12)}\\
&\equiv_0 rtg(r) +_b (0 +_c rtg(s)) && \text{(G3', BA)}\\
&\equiv_0 rtg(r) +_b rtg(s) && \text{(Lemma 3.3.12)}\\
rtg(r * s) &= rtg(r)^{(c)}rtg(s)\\
&\equiv_0 rtg(r)(rtg(r)^{(c)}rtg(s)) +_c rtg(s) && \text{(FP1)}\\
&\equiv_0 (rtg(r) +_b 0)(rtg(r)^{(c)}rtg(s)) +_c rtg(s) && \text{(Lemma 3.3.12)}\\
&\equiv_0 (rtg(r)(rtg(r)^{(c)}rtg(s)) +_b 0) +_c rtg(s) && \text{(G8,G6)}\\
&\equiv_0 rtg(r)(rtg(r)^{(c)}rtg(s)) +_b (0 +_c rtg(s)) && \text{(G3')}\\
&\equiv_0 rtg(r)(rtg(r)^{(c)}rtg(s)) +_b rtg(s) && \text{(Lemma 3.3.12)}\\
&= rtg(r)rtg(r * s) +_b rtg(s)
\end{aligned}$$

$$\equiv_0 rtg(r)^{(b)} rtg(s) \qquad \text{(RSP)}$$

□

The most important property of *rtg* for the completeness proof is that it preserves provable equivalence. This is the content of the following theorem.

**Theorem 3.3.13.** *Let $r_1, r_2 \in Det$. If $r_1 \equiv_* r_2$, then $rtg(r_1) \equiv_0 rtg(r_2)$.*

An intuitive approach to proving Theorem 3.3.13 proceeds by induction on the derivation of $\mathsf{1fMil} \vdash r = s$. However, if one-free regular expressions appear in the derivation of $\mathsf{1fMil} \vdash r = s$ that are not deterministic, then the induction hypothesis cannot be applied: The translation map *rtg* is only defined on *Det*. In other words, the induction hypothesis must be strengthened: Our proof by induction on derivations can only go through if whenever $r, s \in Det$ and $r \equiv_* s$, there is a derivation of $\mathsf{1fMil} \vdash r = s$ in which only deterministic one-free regular expressions appear. Such a derivation is what we call a *deterministic proof*.

**Definition 3.3.14.** Given $r, s \in Det$, we call a proof of $\mathsf{1fMil} \vdash r = s$ a *deterministic proof* if every expression that appears in the proof is a deterministic one-free regular expression (i.e., is in *Det*). We write $\mathsf{1fMil} \vdash_{\text{det}} r = s$ if there is a deterministic proof of $\mathsf{1fMil} \vdash r = s$ and say that $r$ is *deterministic provably equivalent* to $s$.

As it turns out, for deterministic one-free regular expressions $r, s \in Det$, $r \equiv_* s$ if and only if there is a deterministic proof of $\mathsf{1fMil} \vdash r = s$.

**Theorem 3.3.15.** *For any $r, s \in Det$, if $\mathsf{1fMil} \vdash r = s$, then $\mathsf{1fMil} \vdash_{\text{det}} r = s$.*

Our next goal is to prove this theorem.

**The proof of Theorem 3.3.15**

In order to show that every provable equivalence between deterministic one-free regular expressions is obtainable from a deterministic proof, we need to take a detour through the completeness proof of Grabmayer and Fokkink [GF20], outlined in Chapter 2, for regular expressions modulo bisimilarity.

Grabmayer and Fokkink's completeness proof revolves around a notion of *solution to an operational model* that is similar to the one we used in the completeness proof for GKAT in Section 3.1. We adapt this notion to deterministic proofs.

**Definition 3.3.16.** Consider a deterministic prechart $(X,\tau)$. Let $\varphi\colon X \to Det$. We say that $\varphi$ is a *deterministic solution* if for any $x \in X$,

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi(x) = \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{x \xrightarrow{\alpha p} y} \alpha p\ \varphi(y)$$

Two deterministic solutions $\varphi_1$ and $\varphi_2$ are *deterministic provably equivalent* if for any $x \in X$, $\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_1(x) = \varphi_2(x)$.

In Section 2.4, we saw a sufficient condition for guaranteeing that a prechart admits a unique solution up to (not necessarily deterministic) provable equivalence. The condition is the existence of a so-called *layering witness*, an auxiliary transition system with edges labelled either by $\to_{\mathsf{e}}$, for *loop entry*, or $\to_{\mathsf{b}}$, for *body* (see Definition 2.4.2 for details). Furthermore, the unique solution to a well-layered prechart is given by a formula, which we recall next.

**Definition 3.3.17.** Define the two quantities below for any $x \in X$

$$|x|_{en} = \max\{n \mid (\exists x_1,\dots,x_n)\ x \curvearrowright x_1 \curvearrowright \cdots \curvearrowright x_n\}$$
$$|x|_{bo} = \max\{n \mid (\exists x_1,\dots,x_n)\ x \to_{\mathsf{b}} x_1 \to_{\mathsf{b}} \cdots \to_{\mathsf{b}} x_n\}$$

These are finite because in a (locally) finite layering witness $(X,\tau^\bullet)$, $(X,\curvearrowright)$ (see pg. 57), and $(X,\to_{\mathsf{b}})$ contain no infinite paths.

Let $x,z \in X$, $a = \bigvee\{\alpha \mid x \xrightarrow{\alpha p}_{\mathsf{e}}\}$, $b = \bigvee\{\alpha \mid x \xrightarrow{\alpha p} x\}$, and $c = \bigvee\{\alpha \mid x \xrightarrow{\alpha p} \checkmark\}$. The *canonical solution* $\varphi_X$ to $(X,\tau)$ (given the layering witness $(X,\tau^\bullet)$) is defined recursively on $|x|_{bo}$ as follows:

$$\varphi_X(x) = \Big(\sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ x \neq y}} \alpha p\ t_X(y,x)\Big) *^a \Big(\sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^c \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{b}} y \\ x \neq y}} \alpha p\ \varphi_X(y)\Big)$$

where we write $r +^b s$ and $r *^b s$ to denote the terms $r+s$ and $r*s$ respectively, as well as the statement $r \perp_b s$. Above, the expression $t_X(y,x)$ is defined for every pair $(y,x)$ such that $x \curvearrowright y$ by recursion on $(|x|_{en}, |y|_{bo})$ in the lexicographical ordering of $\mathbb{N} \times \mathbb{N}$

as follows: where $a' = \bigvee\{\alpha \mid y \xrightarrow{\alpha p}_{\mathsf{e}}\}$, $b' = \bigvee\{\alpha \mid y \xrightarrow{\alpha p} y\}$, and $c' = \bigvee\{\alpha \mid y \xrightarrow{\alpha p}_{\mathsf{b}} x\}$,

$$t_X(y,x) = \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p + ^{b'} \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ z \neq y}} \alpha p \; t_X(z,y) \Big) *^{a'} \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} x} \alpha p + ^{c'} \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{b}} z \\ z \neq x}} \alpha p \; t_X(z,x) \Big)$$

*Remark* 3.3.18. The lexicographical ordering of $\mathbb{N} \times \mathbb{N}$ is the least order such that $(n,m) \leq (n,k)$ if $m \leq k$ and $(n,m) \leq (l,k)$ if $n \leq l$ for any $n,m,l,k \in \mathbb{N}$. This defines a well-ordering of $\mathbb{N} \times \mathbb{N}$.

We are specifically interested in solving *deterministic* precharts. The following terminology will be useful in proofs.

**Definition 3.3.19.** A state $x$ in a prechart $(X,\tau)$ is *operationally deterministic* if $\tau(x)$ is graph-like, i.e., for any $\xi,\xi' \in \checkmark + X$, $x \xrightarrow{\alpha p} \xi$ and $x \xrightarrow{\alpha q} \xi'$ implies $p = q$ and $\xi = \xi'$.

Thus, a prechart is deterministic if and only if every of its states is operationally deterministic. Determinism is preserved by homomorphisms.

**Lemma 3.3.20.** *If $x$ is an operationally deterministic state of the prechart $(X,\tau_X)$ and $h\colon (X,\tau_X) \to (Y,\tau_Y)$ is a homomorphism, then $h(x)$ is operationally deterministic.*

*Proof.* There are three cases to consider.

1. Suppose $h(x) \xrightarrow{\alpha p} \checkmark$ and $h(x) \xrightarrow{\alpha q} \checkmark$. Then $(\alpha p, \checkmark), (\alpha q, \checkmark) \in \tau_X(x)$ because $h$ is a homomorphism. It follows that $p = q$, because $x$ is operationally deterministic.
2. If $h(x) \xrightarrow{\alpha p} y_1$ and $h(x) \xrightarrow{\alpha q} y_2$ for some $y,y' \in Y$, then there are $x_1,x_2 \in X$ such that $h(x_1) = y_1$, $h(x_2) = y_2$, $x \xrightarrow{\alpha p} x_1$, and $x \xrightarrow{\alpha q} x_2$. Since $x$ is operationally deterministic, $p = q$ and $x_1 = x_2$. Hence, $h(x_1) = h(x_2)$.
3. If $h(x) \xrightarrow{\alpha p} \checkmark$ and $h(x) \xrightarrow{\alpha q} y$ for some $y \in Y$, then $x \xrightarrow{\alpha p} \checkmark$ and $x \xrightarrow{\alpha q} x'$ for some $x'$ such that $h(x') = y$. This is not possible because $x$ is operationally deterministic, which would then require $p = q$ and $\checkmark = x'$, despite $X \cap \checkmark = \emptyset$. □

We are now ready to describe the structure of the proof of Theorem 3.3.15. The proof requires the following four facts about deterministic regular expressions, deterministic well-layered precharts, and their deterministic solutions.

**Fact 1** (Lemma 3.3.21) *Det* is a deterministic subcoalgebra of $(\mathit{StExp},\ell)$.

**Fact 2** (Lemma 3.3.23) For any $r \in \mathit{Det}$, the inclusion map $\mathrm{in}_{\langle r \rangle} \colon \langle r \rangle \to \mathit{StExp}$ is a deterministic solution. This is equivalent to saying that

$$\mathsf{1fMil} \vdash_{\det} r = \sum_{r \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{r \xrightarrow{\alpha p} s} \alpha p \; s$$

**Fact 3** (Theorem 3.3.26) Fix a layering witness $(X, \tau^\bullet)$ for a deterministic prechart $(X, \tau)$ (Definition 2.4.2).

(a) $\varphi_X$ is a deterministic solution to $(X, \tau)$. That is, for any $x \in X$,

$$\mathsf{1fMil} \vdash_{\det} \varphi_X(x) = \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{x \xrightarrow{\alpha p} y} \alpha p \; \varphi_X(y)$$

(b) For any deterministic solution $\psi$ to $(X, \tau)$ and for all $x \in X$,

$$\mathsf{1fMil} \vdash_{\det} \varphi_X(x) = \psi(x)$$

**Fact 4** (Lemma 3.3.27) Let $h \colon (X, \tau_X) \to (Y, \tau_Y)$ be a homomorphism between deterministic precharts and let $\varphi \colon Y \to \mathit{Det}$ be a deterministic solution to $(Y, \tau)$. Then $\varphi \circ h$ is a deterministic solution to $(X, \tau_X)$.

We sketch the proof of Theorem 3.3.15 below, showing how these four facts collectively imply Theorem 3.3.15.

*Proof of Theorem 3.3.15.* Suppose $r, s \in \mathit{Det}$. By Lemma 3.3.21 (Fact 1), $\langle r \rangle$ and $\langle s \rangle$ are deterministic well-layered precharts, because they are subcoalegbras of $(\mathit{StExp}, \ell)$ contained in $\mathit{Det}$ and every subcoalgebra of a well-layered prechart is well-layered.

Now, if $\mathsf{1fMil} \vdash r = s$, then $r \leftrightarrows s$ by soundness. This means there is a minimal prechart $(X, \tau)$ and two quotient homomorphisms $\langle r \rangle \xrightarrow{h} (X, \tau) \xleftarrow{k} \langle s \rangle$ such that $h(r) = k(s)$. Since well-layeredness and determinism are preserved by homomorphisms (Theorem 2.4.12 and Lemma 3.3.20), $(X, \tau)$ is a deterministic well-layered prechart. It follows from Theorem 3.3.26 (Fact 3) that $(X, \tau)$ has a deterministic solution $\varphi_X$.

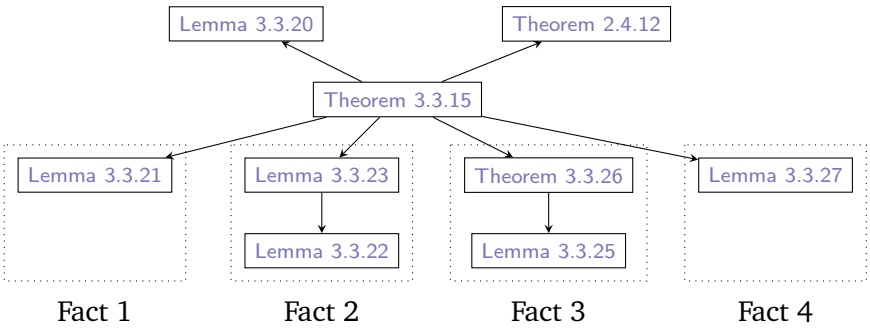
By Lemma 3.3.27 (Fact 4), $\varphi_X \circ h$ and $\varphi_X \circ k$ are deterministic solutions to $\langle r \rangle$ and $\langle s \rangle$ respectively. Lemmas 3.3.21 and 3.3.23 (Facts 1 and 2) tell us that $\mathrm{in}_{\langle r \rangle}$ and $\mathrm{in}_{\langle s \rangle}$ are also deterministic solutions to $\langle r \rangle$ and $\langle s \rangle$ respectively. Therefore, by

Theorem 3.3.26 (Fact 3),

$$\mathsf{1fMil} \vdash_{\det} r = \mathrm{in}_{\langle r \rangle}(r) = \varphi_X \circ h(r)$$
$$\mathsf{1fMil} \vdash_{\det} s = \mathrm{in}_{\langle s \rangle}(s) = \varphi_X \circ k(s)$$

Since $h(r) = k(s)$, we see from the derivations above that $\mathsf{1fMil} \vdash_{\det} r = s$. □

The proof of Theorem 3.3.15 required four facts, which we record as Lemmas 3.3.21, 3.3.23 and 3.3.27 and Theorem 3.3.26. We prove each of these individually, although two of them require further lemmas, and one a further definition. The dependency graph of the results that follow can be drawn like so: Where $A \to B$ denotes that $B$ is used in the proof of $A$, we have




Let us start by proving Fact 1.

**Lemma 3.3.21** (Fact 1)**.** *Det is a deterministic subcoalgebra of* $(StExp, \ell)$.

*Proof.* We show by induction on $r \in Det$ that $r$ is operationally deterministic and that if $r \xrightarrow{\alpha p} s$, then $s \in Det$. There are two base cases.

- The deterministic expression $0$ has no outgoing transitions at all, and so the two properties we are trying to show vacuously hold.
- The expression $\alpha p$ has one outgoing transition, $\alpha p \xrightarrow{\alpha p} \checkmark$, which does not land on another state. Every state with at most one outgoing transition is operationally deterministic, so we are done here.

For the inductive step, suppose $r_1, r_2 \in Det$, $r_1, r_2$ are both operationally deterministic, and $r_i \xrightarrow{\alpha p} s$ implies that $s \in Det$ for either $i = 1, 2$.

- Suppose $r_1 \perp_b r_2$ for some $b \in BExp$. If $r_1 + r_2 \xrightarrow{\alpha p} \xi$ and $r_1 + r_2 \xrightarrow{\alpha q} \xi'$ for $\alpha \leq b$, then $r_1 \xrightarrow{\alpha p} \xi$ and $r_1 \xrightarrow{\alpha q} \xi'$, because $\bar{b} \cdot r_2 \leftrightarrows 0$. By the induction hypothesis,

$p = q$ and $\xi = \xi'$. Also by the induction hypothesis, if $\xi \in StExp$, then $\xi \in Det$. Similarly for $\alpha \leq \bar{b}$ and $r_2$.

- Now suppose we have $r_1 r_2 \xrightarrow{\alpha p} s$ and $r_1 r_2 \xrightarrow{\alpha q} s'$ (it is not possible for $r_1 r_2 \to \checkmark$). Then $r_1 \xrightarrow{\alpha p}$ and $r_1 \xrightarrow{\alpha q}$ are two outgoing transitions from $r_1$ (we leave the targets of the transitions out for now). Because $r_1$ is operationally deterministic, $p = q$. Now, if $r_1 \xrightarrow{\alpha p} \checkmark$, then $r_1 \xrightarrow{\alpha p} s$ and $r_1 \xrightarrow{\alpha p} s'$. By operational determinism, $s = s'$, and by the induction hypothesis, $s \in Det$. If $r_1 \xrightarrow{\alpha p} t$, then $s = t r_2 = s'$. Furthermore, by the induction hypothesis, $t \in Det$, so $t r_2 \in Det$ as desired.
- Now suppose $r_1 \perp_b r_2$ for some $b \in BExp$. If $r_1 * r_2 \xrightarrow{\alpha p} \xi$ and $r_1 * r_2 \xrightarrow{\alpha q} \xi'$ for some $\alpha \leq \bar{b}$, then $r_2 \xrightarrow{\alpha p} \xi$ and $r_2 \xrightarrow{\alpha q} \xi'$, because $\bar{b} \cdot r_1 \leftrightarrows 0$. By the induction hypothesis, $p = q$ and $\xi = \xi' \in \checkmark + Det$. Now suppose $\alpha \leq b$. Then $r_1 \xrightarrow{\alpha p} s$ and $r_1 \xrightarrow{\alpha q} s'$ and $\xi = s(r_1 * r_2)$ and $\xi' = s'(r_1 * r_2)$. By the induction hypothesis applied to $r_1$, $p = q$ and $s = s' \in Det$. Hence, $s(r_1 * r_2) = s'(r_1 * r_2) \in Det$. □

Now we turn our attention to Fact 2. It is important to the proof of Fact 2 that deterministic provable equivalence is a congruence relation on deterministic one-free regular expressions.

**Lemma 3.3.22.** *Let $r_1, r_2, s_1, s_2 \in Det$. If $\mathsf{1fMil} \vdash_{\det} r_i = s_i$ for $i = 1, 2$, then*

1. $\mathsf{1fMil} \vdash_{\det} r_1 + r_2 = s_1 + s_2$ *whenever* $r_1 \perp r_2$ *and* $s_1 \perp s_2$
2. $\mathsf{1fMil} \vdash_{\det} r_1 r_2 = s_1 s_2$
3. $\mathsf{1fMil} \vdash_{\det} r_1 * r_2 = s_1 * s_2$ *whenever* $r_1 \perp r_2$ *and* $s_1 \perp s_2$

*Proof.* In each case, construct deterministic proofs of $r_1 = r_2$ and $s_1 = s_2$. Then complete the proofs of $\mathsf{1fMil} \vdash r_1 + r_2 = s_1 + s_2$, $\mathsf{1fMil} \vdash r_1 r_2 = s_1 s_2$, and $\mathsf{1fMil} \vdash r_1 * r_2 = s_1 * s_2$ with applications of their corresponding congruence rules. □

**Lemma 3.3.23** (Fact 2)**.** *For any $r \in Det$, $\mathrm{in}_{\langle r \rangle} : \langle r \rangle \to StExp$ is a deterministic solution.*

In the proof of this lemma below, we make use of the following notation.

**Definition 3.3.24.** Given $r, s \in StExp$, we write $r +^b s$ and $r *^b s$ for the expressions $r + s$ and $r * s$ respectively, as well as the statement $r \perp_b s$. We refer to these superscripts as *separation markers*.

In particular, if $r, s \in Det$, then $r +^b s$ and $r *^b s$ indicate that $r + s$ and $r * s$ are deterministic expressions as well.

*Proof of Lemma 3.3.23.* We are going to prove that

$$\mathsf{1fMil} \vdash_{\mathrm{det}} r = \sum_{r \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{r \xrightarrow{\alpha p} s} \alpha p s$$

by induction on $r \in \mathit{Det}$. We have yet to see why this equation is between two deterministic expressions, however: Notice that on the right-hand side of the equation above, $\sum_{r \xrightarrow{\alpha p} \checkmark} \alpha p$ and $\sum_{r \xrightarrow{\alpha p} s} \alpha p s$ are deterministic expressions because there is at most one transition $r \xrightarrow{\alpha p}$ per $\alpha \in \mathit{At}$ by operational determinism of $r$. Moreover, if $b = \bigvee\{\alpha \mid r \xrightarrow{\alpha p} \checkmark\}$, then $\sum_{r \xrightarrow{\alpha p} \checkmark} \alpha p \perp_b \sum_{r \xrightarrow{\alpha p} s} \alpha p s$. This shows that the right-hand side of the equation we are about to derive is deterministic.

There are two base cases.

- Since $0$ has no outgoing transitions,

$$\mathsf{1fMil} \vdash \sum_{0 \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{0 \xrightarrow{\alpha p} s} \alpha p s = 0 + 0 = 0$$

  where the first equality is literal syntactic equality and the second is the axiom $x + x = x$. We have already seen that the first expression is deterministic. The expression $0 + 0$ is deterministic because $0 \in \mathit{Det}$ and $0 \perp_1 0$.
- Since $\alpha p$ has only one outgoing transition,

$$\mathsf{1fMil} \vdash \sum_{\alpha p \xrightarrow{\beta q} \checkmark} \beta q + \sum_{\alpha p \xrightarrow{\beta q} s} \beta q s = \alpha p + 0 = \alpha p$$

  We have already seen that the first expression is deterministic. The second is deterministic because $0, \alpha p \in \mathit{Det}$ and $\alpha p \perp_\alpha 0$.

For the inductive step, assume that for $i = 1, 2$,

$$\mathsf{1fMil} \vdash_{\mathrm{det}} r_i = \sum_{r_i \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{r_i \xrightarrow{\alpha p} s} \alpha p s$$

- Let $r_1 \perp_c r_2$, and define $b_i = \bigvee\{r_i \xrightarrow{\alpha p} \checkmark\}$ for $i = 1, 2$. Then

  $\mathsf{1fMil} \vdash_{\mathrm{det}} r_1 +^c r_2$

$$= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s \Big) +^{c} \Big( \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big)$$

(Lemma 3.3.22, ind. hyp.)

$$= \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1 \wedge c} \Big( \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s +^{c} \Big( \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) \Big) \qquad \text{(assoc.)}$$

$$= \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1 \wedge c} \Big( \Big( \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s +^{c} \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p \Big) +^{c \vee b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) \qquad \text{(assoc.)}$$

$$= \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1 \wedge c} \Big( \Big( \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{\bar{c}} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s \Big) +^{c \vee b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big)$$

(comm., $r \perp_c s$ iff $s \perp_{\bar{c}} r$)

$$= \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1 \wedge c} \Big( \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{\bar{c} \wedge b_2} \Big( \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s +^{c \vee b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) \Big) \qquad \text{(assoc.)}$$

$$= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1 \wedge c} \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p \Big) +^{b} \Big( \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s +^{c \vee b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) \qquad \text{(assoc.,†)}$$

$$= \sum_{r_1 + r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b} \sum_{r_1 + r_2 \xrightarrow{\alpha p} s} \alpha p s$$

†In the last two steps, we define $b = (b_1 \wedge c) \vee (\bar{c} \wedge b_2)$. The separation markers in the derivation above indicate how to construct each expression that appears as a deterministic expression. This establishes that there is a deterministic proof of $\mathsf{1fMil} \vdash r_1 + r_2 = \sum_{r_1+r_2 \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{r_1+r_2 \xrightarrow{\alpha p} s} \alpha p s$.

- In the sequential composition inductive case, we have

$$\begin{aligned}
\mathsf{1fMil} \vdash_{\mathrm{det}} r_1 r_2 &= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s \Big) r_2 && \text{(Lemma 3.3.22, ind. hyp.)} \\
&= \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p r_2 +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s r_2 && \text{(dist.)} \\
&= 0 +^{0} \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p r_2 +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s r_2 \Big) && (\neg(r_1 r_2 \xrightarrow{\alpha p} \checkmark)) \\
&= \sum_{r_1 r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{0} \sum_{r_1 r_2 \xrightarrow{\alpha p} s} \alpha p s && \text{(rearr. summands)}
\end{aligned}$$

- In the star inductive case, let $r_1 \perp_c r_2$ and compute

$$\mathsf{1fMil} \vdash_{\mathrm{det}} r_1 * r_2$$

$$
\begin{aligned}
&= r_1(r_1 * r_2) +^c r_2 && \text{(FP1)}\\
&= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s \Big)(r_1 * r_2) +^c r_2 && \text{(Lemma 3.3.22, ind. hyp.)}\\
&= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p (r_1 * r_2) +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s (r_1 * r_2) \Big) +^c r_2 && \text{(dist.)}\\
&= \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p (r_1 * r_2) +^{b_1} \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s (r_1 * r_2) \Big)\\
&\qquad +^c \Big( \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b_2} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) && \text{(Lemma 3.3.22, ind. hyp.)}\\
&= \sum_{r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{\bar{c} \wedge b_2} \Big( \sum_{r_1 \xrightarrow{\alpha p} \checkmark} \alpha p (r_1 * r_2)\\
&\qquad +^{b_1 \wedge c} \Big( \sum_{r_1 \xrightarrow{\alpha p} s} \alpha p s (r_1 * r_2) +^{c \wedge \bar{b_1}} \sum_{r_2 \xrightarrow{\alpha p} s} \alpha p s \Big) \Big) && \text{(assoc.)}\\
&= \sum_{r_1 * r_2 \xrightarrow{\alpha p} \checkmark} \alpha p +^{b} \sum_{r_1 * r_2 \xrightarrow{\alpha p} s} \alpha p s
\end{aligned}
$$

where $b = \bar{c} \vee b_1$. □

Now we turn our attention to Fact 3. Fix a locally finite layering witness $(X, \tau^\bullet)$ for a deterministic prechart $(X, \tau)$. Write $\varphi_X$ for the canonical solution to $(X, \tau)$ given $(X, \tau^\bullet)$. We need the following lemma to prove Fact 3.

**Lemma 3.3.25.** *Let $x, z \in X$.*

1. *$\varphi_X(x)$ and $t_X(x,z)$ are deterministic expressions.*
2. *If $x \curvearrowright y$, then*
$$\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_X(y) = t_X(y,x)\varphi_X(y)$$
3. *If $x \curvearrowright y$ and $\psi$ is any deterministic solution to $(X, \tau)$, then*
$$\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(y) = t_X(y,x)\psi(y)$$

*Proof.* We begin by showing item 1, that $\varphi_X(x)$ and $t_X(x,y)$ are deterministic expressions for any $x, z \in X$. This can be seen by induction on $|x|_{bo}$ and $(|x|_{en}, |y|_{bo})$ in $\mathbb{N}$ and the lexicographical ordering of $\mathbb{N} \times \mathbb{N}$ respectively. In the base cases, $|x|_{bo} = 0$ in the first case and $|x|_{en} = 1, |y|_{bo} = 0$ in the second. Note that $|x|_{en} = 1$ means that $|y|_{en} = 0$

in the second case.

$$\varphi_X(x) = \Big( \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p +^b \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p \; t_X(y,x) \Big) *^a \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^c 0 \Big)$$
$$t_X(y,x) = \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^{b'} 0 \Big) *^{a'} \Big( 0 +^{c'} 0 \Big)$$

These are both deterministic, as seen from the separation markers $a,b,c$ and $a',b',c'$, and because $x$ is operationally deterministic.

For the inductive step, observe from the separation markers in Definition 3.3.17 that $\varphi_X(x)$ and $t_X(y,x)$ are constructed from sums and stars of separated one-free regular expressions. It therefore suffices to see that the sub-expressions $t_X(z,y), \varphi_X(y), t_X(z,x)$ are all deterministic. This follows from the induction hypothesis:

- Where $t_X(z,y)$ appears in $t_X(y,x)$, $x \curvearrowright y$. This means that $|y|_{en} < |x|_{en}$ and therefore $(|y|_{en}, |z|_{bo}) < (|x|_{en}, |y|_{bo})$ in the lexicographical ordering. By the induction hypothesis, $t_X(z,y)$ is deterministic.
- Where $\varphi_X(y)$ appears in $\varphi_X(x)$, $x \to_{\mathsf{b}} y$. This means that $|y|_{bo} < |x|_{bo}$. By the induction hypothesis, $\varphi_X(y)$ is deterministic.
- Where $t_X(z,x)$ appears in $t_X(y,x)$, $y \to_{\mathsf{b}} z$. This means that $|z|_{bo} < |y|_{bo}$, so that $(|x|_{en}, |z|_{bo}) < (|x|_{en}, |y|_{bo})$ in the lexicographical ordering. By the induction hypothesis, $t_X(z,x)$ is deterministic.

Thus, $\varphi_X(x)$ and $t_X(y,x)$ are both deterministic.

We show item 2 by induction on $|y|_{bo}$. Let $b = \bigvee\{\alpha \mid y \xrightarrow{\alpha p} y\}$, let

$$r = \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p \; t_X(z,y)$$

and $a = \bigvee\{\alpha \mid y \xrightarrow{\alpha p}_{\mathsf{b}} x\}$, $c = \bigvee\{\alpha \mid (\exists z)\; y \xrightarrow{\alpha p}_{\mathsf{e}} z\}$, and $d = \bigvee\{\alpha \mid y \xrightarrow{\alpha p} \checkmark\}$. Note that we have already seen that $r$ is deterministic. In the base case, $|y|_{bo} = 0$, $t_X(y,x) = r * 0$, so we have

$$\mathsf{1fMil} \vdash_{\mathrm{det}} (r*0)\varphi_X(x) = r(r*0)\varphi_X(x) +^c 0\varphi_X(x) = r(r*0)\varphi_X(x) +^c 0 \overset{(\mathrm{RSP})}{=} r*0$$

In the inductive step, we have

$$\begin{aligned}
\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_X(y) &= r *^c \Big( \sum_{y \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p \ \varphi_X(z) \Big) && \text{(def)} \\
&= r *^c \Big( 0 +^0 \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p \ \varphi_X(z) \Big) && (\neg(y \to \checkmark)) \\
&= r *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p \ \varphi_X(z) \Big) && \text{(zero)} \\
&= r *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} x} \alpha p \ \varphi_X(x) +^a \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{b}} z \\ z \neq x}} \alpha p \ \varphi_X(z) \Big) && \text{(rearr. summands)} \\
&= r *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} x} \alpha p \ \varphi_X(x) +^a \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{b}} z \\ z \neq x}} \alpha p \ t_X(z,x) \varphi_X(x) \Big) \\
& && \text{(ind. hyp., } |z|_{bo} < |y|_{bo}) \\
&= r *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} x} \alpha p +^a \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{b}} z \\ z \neq x}} \alpha p \ t_X(z,x) \Big) \varphi_X(x) && \text{(dist.)} \\
&= t_X(y,x) \varphi_X(x) && \text{(def. } t_X(y,x))
\end{aligned}$$

Indeed, by item 1 and as indicated by the separation markers above, each of the expressions in this derivation is deterministic.

Finally, we prove item 3 by induction on $(|x|_{en}, |y|_{bo})$ in the lexicographical ordering of $\mathbb{N} \times \mathbb{N}$, assuming $x \curvearrowright y$. In the base case, $|x|_{en} = 1$ and $|y|_{bo} = 0$, and we also know $|y|_{en} = 0$ since $x \curvearrowright y$. Since $x \curvearrowright y$, $\neg(y \to \checkmark)$, so

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(y) = 0 +^0 \sum_{y \xrightarrow{\alpha p} z} \alpha p \ \psi(z) = \sum_{y \xrightarrow{\alpha p} z} \alpha p \ \psi(z) \qquad (\neg(y \to \checkmark), \text{ zero})$$

Let $c = \bigvee\{\alpha \mid (\exists z)\ y \xrightarrow{\alpha p}_{\mathsf{e}} z\}$ and $b = \bigvee\{\alpha \mid y \xrightarrow{\alpha p} y\}$. Then

$$\begin{aligned}
\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(y) &= \sum_{y \xrightarrow{\alpha p} z} \alpha p \ \psi(z) \\
&= \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} z} \alpha p \ \psi(z) +^c \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p \ \psi(z) && \text{(rearr. summands)}
\end{aligned}$$

$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p\ \psi(y) +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ \psi(z) \Big) +^c \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ \psi(z)$$

(rearr. summands)

In the base case, the above becomes

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p\ \psi(y) +^b 0 \Big) +^c 0 = \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p\ \psi(y) +^b 0\ \psi(y) \Big) +^c 0$$
$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b 0 \Big) \psi(y) +^c 0$$

so by (RSP),

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(y) = (\sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b 0) *^c 0 = \Big( (\sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b 0) *^c 0 \Big) \psi(x) = t_X(y,x)\psi(x)$$

In the inductive step,

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(y) = \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p\ \psi(y) +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ \psi(z) \Big) +^c \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ \psi(z)$$
$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p\ \psi(y) +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ t_X(z,y)\psi(y) \Big) +^c \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ t_X(z,x)\psi(x)$$

(ind. hyp. with $|y|_{en} < |x|_{en}$ and ind. hyp with $|z|_{bo} < |y|_{bo}$)

$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ t_X(z,y) \Big) \psi(y) +^c \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ t_X(z,x)\psi(x)$$

(dist.)

$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ t_X(z,y) \Big) *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ t_X(z,x)\psi(x) \Big) \quad \text{(RSP)}$$
$$= \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{e}} y} \alpha p +^b \sum_{\substack{y \xrightarrow{\alpha p}_{\mathsf{e}} z \\ y \neq z}} \alpha p\ t_X(z,y) \Big) *^c \Big( \sum_{y \xrightarrow{\alpha p}_{\mathsf{b}} z} \alpha p\ t_X(z,x) \Big) \psi(x)$$

(assoc.)

$$= t_X(y,x)\psi(x) \qquad \square$$

**Theorem 3.3.26** (Fact 3)**.** *The canonical solution $\varphi_X$ is a deterministic solution to*

$(X,\tau)$. *That is, for any* $x \in X$,

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_X(x) = \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{x \xrightarrow{\alpha p} y} \alpha p \; \varphi_X(y)$$

*Furthermore, for any deterministic solution* $\psi$ *to* $(X,\tau)$ *and any* $x \in X$,

$$\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_X(x) = \psi(x)$$

*Proof of Theorem 3.3.26.* To see that the canonical solution to $(X,\tau)$ is a deterministic solution, let $x,z \in X$, and $b = \bigvee\{\alpha \mid x \xrightarrow{\alpha p} x\}$, $c = \bigvee\{\alpha \mid x \xrightarrow{\alpha p}_{\mathsf{e}}\}$, and $d = \bigvee\{\alpha \mid x \xrightarrow{\alpha p} \checkmark\}$. We derive

$$\begin{aligned}
&\mathsf{1fMil} \vdash_{\mathrm{det}} \varphi_X(x) \\
&= \Big( \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ x \neq y}} \alpha p \; t_X(y,x) \Big) *^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{b}} y \\ x \neq y}} \alpha p \; \varphi_X(y) \Big) && \text{(def.)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ x \neq y}} \alpha p \; t_X(y,x) \Big) \varphi_X(x) +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{b}} y \\ x \neq y}} \alpha p \; \varphi_X(y) \Big) && \text{(FP1)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p \; \varphi_X(x) +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ x \neq y}} \alpha p \; t_X(y,x) \varphi_X(x) \Big) \\
&\qquad +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{b}} y \\ x \neq y}} \alpha p \; \varphi_X(y) \Big) && \text{(dist.)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p}_{\mathsf{e}} x} \alpha p \; \varphi_X(x) +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ x \neq y}} \alpha p \; \varphi_X(y) \Big) +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{b}} y \\ x \neq y}} \alpha p \; \varphi_X(y) \Big) \\
&\qquad && \text{(Lemma 3.3.25 item 2)} \\
&= \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p} y} \alpha p \; \varphi_X(y) && \text{(rearr. summands)}
\end{aligned}$$

In the last step, we used the fact that $\bar{d} \geq_{\mathsf{BA}} c \geq_{\mathsf{BA}} b$.

Now let $\psi$ be another deterministic solution to $(X,\tau)$. To see that $\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(x) = \varphi_X(x)$ for any $x \in X$, we proceed by induction on $|x|_{bo}$. The inductive step

proceeds as follows:

$$
\begin{aligned}
&\mathsf{1fMil} \vdash_{\mathrm{det}} \psi(x) \\
&= \Big( \sum_{x \xrightarrow{\alpha p} x} \alpha p \ \psi(x) + ^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ y \neq x}} \alpha p \ \psi(y) \Big) +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y} \alpha p \ \psi(y) \Big) && \text{(def.)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p} x} \alpha p \ \psi(x) + ^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ y \neq x}} \alpha p \ t_X(y,x) \psi(x) \Big) +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y} \alpha p \ \psi(y) \Big) \\
& && \text{(Lemma 3.3.25 item 3)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ y \neq x}} \alpha p \ t_X(y,x) \Big) \psi(x) +^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y} \alpha p \ \psi(y) \Big) && \text{(dist.)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ y \neq x}} \alpha p \ t_X(y,x) \Big) *^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y} \alpha p \ \psi(y) \Big) && \text{(RSP)} \\
&= \Big( \sum_{x \xrightarrow{\alpha p} x} \alpha p +^b \sum_{\substack{x \xrightarrow{\alpha p}_{\mathsf{e}} y \\ y \neq x}} \alpha p \ t_X(y,x) \Big) *^c \Big( \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p +^d \sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y} \alpha p \ \varphi_X(y) \Big) \\
& && \text{(ind. hyp., } |y|_{bo} < |x|_{bo}\text{)} \\
&= \varphi_X(x)
\end{aligned}
$$

The base case is similar, except that it skips the second to last equality above, because $\sum_{x \xrightarrow{\alpha p}_{\mathsf{b}} y}$ is an empty sum. □

Finally, we prove Fact 4.

**Lemma 3.3.27** (Fact 4)**.** *Let $h\colon (X,\tau_X) \to (Y,\tau_Y)$ be a homomorphism between deterministic precharts and let $\psi\colon Y \to Det$ be a deterministic solution to $(Y,\tau)$. Then $\psi \circ h$ is a deterministic solution to $(X,\tau_X)$.*

*Proof.* The key observations here are (1) $x \xrightarrow{\alpha p} \checkmark$ iff $h(x) \xrightarrow{\alpha p} \checkmark$, and (2) $h(x) \xrightarrow{\alpha p} y$ iff there is an $x' \in X$ such that $h(x') = y$ and $x \xrightarrow{\alpha p} x'$. Since (1) implies $\sum_{h(x) \xrightarrow{\alpha p} \checkmark} \alpha p = \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p$, and since (2) implies $\sum_{h(x) \xrightarrow{\alpha p} y} \alpha p \ \psi(y) = \sum_{x \xrightarrow{\alpha p} x'} \alpha p \ \psi(h(x'))$,

$$
\begin{aligned}
\mathsf{1fMil} \vdash_{\mathrm{det}} \psi \circ h(x) &= \sum_{h(x) \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{h(x) \xrightarrow{\alpha p} y} \alpha p \ \psi(y) \\
&= \sum_{x \xrightarrow{\alpha p} \checkmark} \alpha p + \sum_{x \xrightarrow{\alpha p} x'} \alpha p \ \psi \circ h(x')
\end{aligned}
$$

Where the second equality is syntax equality. It follows that $\psi \circ h$ is a deterministic solution to $(X, \tau)$. □

This concludes our verification of each of the four facts used in the proof of Theorem 3.3.15.

**The last few steps of the completeness proof**

With Theorem 3.3.15 in hand, we are ready to establish the necessary property of $rtg$ used in the proof of completeness, namely Theorem 3.3.13, which says that $r \equiv_* s$ implies $rtg(r) \equiv_0 rtg(s)$ for any $r, s \in Det$.

*Proof of Theorem 3.3.13.* Let $r, s \in Det$, and suppose $r \equiv_* s$. Then, by Theorem 3.3.15, $\mathsf{1fMil} \vdash_{\text{det}} r = s$. We proceed by induction on the deterministic derivation of $\mathsf{1fMil} \vdash r = s$. In the base case, we verify the one-free star behaviour axioms directly.

**(B0)** Every $r \in Det$ such that $r + r \in Det$ satisfies $r \equiv_* 0$, so by (G0), $rtg(r+r) = rtg(r) +_1 rtg(r) \equiv_0 rtg(r)$.

**(B1)** The maximal test separating $r$ from 0 is 1, so $rtg(r+0) = rtg(r) +_1 rtg(0) \equiv_0 rtg(r)$ by (G0).

**(B2)** If $r \perp_b s$, then $rtg(r+s) \equiv_0 rtg(r) +_b rtg(s) \equiv_0 rtg(s) +_{\bar{b}} rtg(r) \equiv_0 rtg(s+r)$ by (G2) and Lemma 3.3.11 (because $s \perp_{\bar{b}} r$).

**(B3)** Letting $r \perp_b s$ and $(r+s) \perp_c t$, we have $r \perp_{b \wedge c} t$ and $s \perp_c t$ as well, so

$$\begin{aligned}
rtg((r+s)+t) &\equiv_0 (rtg(r) +_b rtg(s)) +_c rtg(t) && \text{(Lemma 3.3.11)} \\
&\equiv_0 rtg(r) +_{b \wedge c} (rtg(s) +_c rtg(t)) && \text{(G3')} \\
&\equiv_0 rtg(r) +_{b \wedge c} rtg(s+t) && \text{(Lemma 3.3.11)} \\
&\equiv_0 rtg(r + (s+t)) && \text{(Lemma 3.3.11)}
\end{aligned}$$

**(B6)** Similarly in the left zero case: By (G6),

$$rtg(0r) = rtg(0) gtr(r) = 0 rtg(r) \equiv_0 0 = rtg(0)$$

**(B7)** No further considerations in the associativity case:

$$\begin{aligned} rtg(r(st)) &= rtg(r)rtg(st) \\ &= rtg(r)(rtg(s)rtg(t)) \\ &\equiv_0 (rtg(r)rtg(s))rtg(t) && \text{(G7)} \\ &= rtg((rs)t) \end{aligned}$$

**(B8)** If $r \perp_b s$, then $rt \perp_b st$, so

$$\begin{aligned} rtg((r+s)t) &= rtg(r+s)rtg(t) \\ &\equiv_0 (rtg(r) +_b rtg(s))rtg(t) && \text{(Lemma 3.3.11)} \\ &\equiv_0 rtg(r)rtg(t) +_b rtg(s)rtg(t) && \text{(G8)} \\ &\equiv_0 rtg(rt+st) && \text{(Lemma 3.3.11)} \end{aligned}$$

**(FP1)** Suppose $r \perp_b s$. Then $r(r*s) \perp_b s$ as well, so

$$\begin{aligned} rtg(r*s) &\equiv_0 rtg(r)^{(b)}rtg(s) && \text{(Lemma 3.3.11)} \\ &\equiv_0 rtg(r)(rtg(r)^{(b)}rtg(s)) +_b rtg(s) && \text{(FP1)} \\ &\equiv_0 rtg(r)rtg(r*s) +_b rtg(s) && \text{(Lemma 3.3.11)} \\ &= rtg(r(r*s)) +_b rtg(s) \\ &\equiv_0 rtg(r(r*s)+s) && \text{(Lemma 3.3.11)} \end{aligned}$$

For the inductive step, assume that $\mathsf{1fMil} \vdash_{\text{det}} r_i = s_i$ implies $rtg(r_i) \equiv_0 rtg(s_i)$ for $i = 1,2$. We consider the congruence rules, (RSP), symmetry (Sym) and transitivity (Tra).

**(+)** Suppose the deterministic proof ends with

$$\frac{r_1 = r_2 \qquad s_1 = s_2}{r_1 + s_1 = r_2 + s_2}$$

and assume that $rtg(r_i) \equiv_0 rtg(s_i)$. We are also assuming that $r_i + s_i$ is deterministic, so we must have $b_1, b_2 \in BExp$ such that $r_i \perp_{b_i} s_i$, $i = 1,2$. However, since $r_1 \equiv_* r_2$ and $s_1 \equiv_* s_2$, $r_1 \perp_b s_1$ if and only if $r_2 \perp_b s_2$, so we may as well take

$b_1 = b_2 = b$. We have

$$rtg(r_1 + s_1) \equiv_0 rtg(r_1) +_b rtg(s_1) \equiv_0 rtg(r_2) +_b rtg(s_2) \equiv_0 rtg(r_2 + s_2)$$

**(∗)** Suppose the deterministic proof ends with

$$\frac{r_1 = s_1 \qquad r_2 = s_2}{r_1 * s_1 = r_2 * s_2}$$

and assume that $rtg(r_i) \equiv_0 rtg(s_i)$, $i = 1,2$. Again, we obtain a common $b \in BExp$ such that $r_i \perp_b s_i$ for $i = 1,2$. Using Lemma 3.3.11, we have

$$rtg(r_1 * s_1) \equiv_0 rtg(r_1)^{(b)} rtg(s_1) \equiv_0 rtg(r_2)^{(b)} rtg(s_2) \equiv_0 rtg(r_2 * s_2)$$

**(RSP)** Suppose the deterministic proof ends with the rule $t = rt + s \Rightarrow t = r * s$ and assume that $rtg(t) \equiv_0 rtg(rt + s)$. Since we are also assuming that $r * s \in Det$, there is a $b \in BExp$ such that $r \perp_b s$. We also have $rt \perp_b s$, so $rtg(t) \equiv_0 rtg(r)rtg(t) +_b rtg(s)$ by Lemma 3.3.11. It follows from (RSP) that

$$rtg(t) \equiv_0 rtg(r)^{(b)} rtg(s) \equiv_0 rtg(r * s)$$

**(Sym)** Suppose the deterministic proof ends with

$$\frac{r_1 = r_2}{r_2 = r_1}$$

Then by symmetry of $\equiv_0$ and the induction hypothesis, $rtg(r_2) \equiv_0 rtg(r_1)$.

**(Tra)** Suppose the deterministic proof ends with

$$\frac{r_1 = s \qquad s = r_2}{r_1 = r_2}$$

Then by assumption, $\mathsf{1fMil} \vdash_{\text{det}} r_1 = s$ and $\mathsf{1fMil} \vdash_{\text{det}} s = r_2$. By the induction hypothesis, $rtg(r_1) \equiv_0 rtg(s) \equiv_0 rtg(r_2)$. This implies that $rtg(r_1) \equiv_0 rtg(r_2)$ by transitivity of $\equiv_0$. □

The last fact needed in the proof of completeness (Theorem 3.2.18) is that, up to provable equivalence, every skip-free GKAT expression is equivalent to its back-translation. This can be proven with the following lemma in hand.

**Lemma 3.3.28.** *If $e \equiv_0 f$, then $gtr(e) \equiv_* gtr(f)$.*

*Proof.* Let $e \equiv_0 f$. By soundness, $e \leftrightarrows f$. By Lemma 3.3.8, $gtr(e) \leftrightarrows gtr(f)$. By completeness of $\equiv_*$ with respect to bisimilarity (Theorem 2.4.13), $gtr(e) \equiv_* gtr(f)$. □

**Lemma 3.3.29.** *For any $e \in GExp_{\mathsf{sf}}$, $e \equiv_0 rtg(gtr(e))$.*

*Proof.* We are going to show, by induction on $e \in GExp_{\mathsf{sf}}$, that for any $d \in BExp$, $e +_d 0 \equiv_0 rtg(gtr(e +_d 0))$. Along the way, we will use the following observations: For $e \in GExp$ and $d \in BExp$,

$$
\begin{aligned}
gtr(e +_d 0) &= d \cdot gtr(e) + \bar{d} \cdot 0 \equiv_* d \cdot gtr(e) \\
rtg(d \cdot gtr(e)) &= rtg\left(\sum_{\alpha \le d} \alpha \cdot gtr(e)\right) && (d \cdot gtr(e) \perp_d \bar{d} \cdot 0, \text{ Lemma 3.3.11}) \\
&\equiv_0 \sum_{\alpha \le d} \alpha \cdot rtg(gtr(e)) +_d 0 && (\text{Lemma 3.3.11, G9 with } c = b = \alpha \le d) \\
&\equiv_0 rtg(gtr(e)) +_d 0 && (\text{G9 with } c = \alpha \le d, b = d)
\end{aligned}
$$

We used the generalized sum notation above, from Remark 3.3.7. Putting the two observations together with Theorem 3.3.13, we see that

$$rtg(gtr(e +_d 0)) \equiv_0 rtg(d \cdot gtr(e)) \equiv_0 rtg(gtr(e)) +_d 0 \qquad (\heartsuit)$$

We will also use Theorem 3.3.13 and Lemma 3.3.28 extensively.

For the base case, we have the following:

- In the 0 case,

$$
\begin{aligned}
rtg(gtr(0 +_d 0)) &\equiv_0 rtg(gtr(0)) +_d 0 && (\heartsuit) \\
&= 0 +_d 0
\end{aligned}
$$

- In the $p \in \Sigma$ case,

$$\begin{aligned}
rtg(gtr(p +_d 0)) &\equiv_0 rtg(gtr(p)) +_d 0 && (\heartsuit)\\
&= rtg\left(\sum_{\alpha\in At} \alpha p\right) +_d 0 \\
&\equiv_0 \sum_{\alpha\in At} \alpha \cdot rtg(\alpha p) +_d 0 && \text{(Lemma 3.3.11)}\\
&\equiv_0 \sum_{\alpha\le d} \alpha \cdot (p +_\alpha 0) +_d 0 && \text{(G10, G2, G0, BA)}\\
&\equiv_0 \sum_{\alpha\le d} \alpha \cdot p +_d 0 && \text{(G9, with } c = b = \alpha)\\
&\equiv_0 p +_d 0 && \text{(G9, with } c = \alpha \le d, b = d)
\end{aligned}$$

Now assume $e +_d 0 \equiv_0 rtg(gtr(e +_d 0))$ and $f +_d 0 \equiv_0 rtg(gtr(f +_d 0))$ for all $d$.

- In the guarded union case,

$$\begin{aligned}
rtg(gtr((e +_b f) +_d 0)) &\equiv_0 rtg(gtr(e +_b f)) +_d 0 && (\heartsuit)\\
&= rtg(b \cdot gtr(e) + \bar{b} \cdot gtr(f)) +_d 0 \\
&\equiv_0 (rtg(gtr(e)) +_b rtg(gtr(f))) +_d 0 && \text{(Lemma 3.3.11)}\\
&\equiv_0 (rtg(gtr(e)) +_d 0) +_b (rtg(gtr(f)) +_d 0) && \text{(BD)}\\
&\equiv_0 (e +_d 0) +_b (f +_d 0) && \text{(ind. hyp.)}\\
&\equiv_0 (e +_b f) +_d 0 && \text{(BD)}
\end{aligned}$$

- For the sequential composition case,

$$\begin{aligned}
rtg(gtr(ef +_d 0)) &\equiv_0 rtg(gtr(ef)) +_d 0 && (\heartsuit)\\
&= rtg(gtr(e))rtg(gtr(f)) +_d 0 \\
&\equiv_0 ef +_d 0 && \text{(ind. hyp.)}
\end{aligned}$$

- For the while loop case, we start with $rtg(gtr(e^{(b)} f)) \equiv_0 e^{(b)} f$. We have

$$\begin{aligned}
rtg(gtr(e^{(b)} f)) &= rtg(b \cdot gtr(e) * \bar{b} \cdot gtr(f)) \\
&\equiv_0 rtg(b \cdot gtr(e))^{(b)} rtg(\bar{b} \cdot gtr(f)) && \text{(Lemma 3.3.11)}
\end{aligned}$$

$$\equiv_0 (rtg(gtr(e)) +_b 0)^{(b)} (rtg(\cdot gtr(f)) +_b 0) \qquad (\heartsuit)$$
$$\equiv_0 (e +_b 0)^{(b)} (0 +_b f) \qquad \text{(ind. hyp.)}$$
$$\equiv_0 e^{(b)} f \qquad \text{(FP2')}$$

Thus,

$$rtg(gtr(e^{(b)} f +_d 0)) \equiv_0 rtg(gtr(e^{(b)} f)) +_d 0 \equiv_0 e^{(b)} f +_d 0$$

This proves the lemma, because taking $d = 1$ we have

$$e \overset{\text{(G0)}}{\equiv_0} e +_1 0 \equiv_0 rtg(gtr(e +_1 0)) \overset{(\heartsuit)}{\equiv_0} rtg(gtr(e)) +_1 0 \overset{\text{(G0)}}{\equiv_0} rtg(gtr(e)) \qquad \square$$

We are now ready to prove Theorem 3.2.18, that skip-free bisimulation GKAT is complete with respect to bisimilarity.

**Theorem 3.2.18** (Completeness I)**.** *Let $e_1, e_2 \in GExp_{\mathsf{sf}}$. If $e_1 \leftrightarrows e_2$, then $e_1 \equiv_0 e_2$.*

*Proof.* Let $e_1, e_2 \in GExp$ be a bisimilar pair of skip-free GKAT expressions. By Lemma 3.3.3, $e_1$ and $e_2$ are bisimilar in $\mathsf{grph}_*(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. By Lemmas 2.2.12 and 3.3.8, the translation $gtr\colon \mathsf{grph}_*(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}}) \to (StExp, \ell)$ preserves bisimilarity, so $gtr(e_1)$ and $gtr(e_2)$ are bisimilar in $(StExp, \ell)$ as well. By Theorem 2.4.13, $gtr(e_1) \equiv_* gtr(e_2)$. Therefore, by Theorem 3.3.13, $rtg(gtr(e_1)) \equiv_0 rtg(gtr(e_2))$. Finally, by Lemma 3.3.29, we have $e_1 \equiv_0 rtg(gtr(e_1)) \equiv_0 rtg(gtr(e_2)) \equiv_0 e_2$. $\square$

## 3.4 Completeness for Skip-free Language GKAT

The previous section establishes that $\equiv_0$-equivalence coincides with bisimilarity for skip-free GKAT expressions by reducing the completeness problem of skip-free GKAT modulo bisimilarity to a solved completeness problem, namely that of one-free star expressions modulo bisimilarity. In this section we prove a completeness result for skip-free GKAT modulo language equivalence by reducing it to the completeness problem of skip-free bisimulation GKAT.

The axiom $e0 = 0$, the only difference between skip-free GKAT and skip-free bisimulation GKAT, indicates that the only semantic difference between bisimilarity and language equivalence in skip-free GKAT is early termination. This motivates our reduction to skip-free GKAT modulo bisimilarity below, which involves reducing each

skip-free expression to an expression representing only the successfully terminating branches of execution.

**Outline of the completeness proof**

Now let us turn to the formal proof of Theorem 3.2.19, which says that if $e, f \in GExp_{\mathsf{sf}}$ are such that $\mathcal{L}(e) = \mathcal{L}(f)$, then $e \equiv f$. Much like our completeness proof for GKAT with the uniqueness axiom (Theorem 3.2.19), our strategy is to produce two terms $\lfloor e \rfloor, \lfloor f \rfloor \in GExp_{\mathsf{sf}}$ such that $e \equiv \lfloor e \rfloor$, $f \equiv \lfloor f \rfloor$ and $\lfloor e \rfloor \leftrightarrows \lfloor f \rfloor$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. The latter property tells us that $\lfloor e \rfloor \equiv_0 \lfloor f \rfloor$ by Theorem 3.2.18, which allows us to conclude $e \equiv f$. The expression $\lfloor e \rfloor$ can be thought of as the *early termination version* of $e$, obtained by pruning the branches of its execution that cannot end in successful termination.

In contrast with the pruning operator (Definition 3.1.32) for GKAT expressions (that required UA), the construction of $\lfloor e_1 \rfloor$ and $\lfloor e_2 \rfloor$ for skip-free $e_1$ and $e_2$ can be computed explicitly from the syntax. To properly define the transformation $\lfloor - \rfloor$ on expressions, we need the notion of a *dead* state in a skip-free automaton, analogous to Definition 3.1.29 [Smo+20].

**Definition 3.4.1.** Let $(X, \delta_X)$ be a skip-free automaton. The set $D(X, \delta_X)$ is the largest subset of $X$ such for all $x \in D(X, \delta_X)$ and $\alpha \in At$, either $\delta_X(x)(\alpha) = \bot$ or $\delta_X(x)(\alpha) \in \Sigma \times D(X, \delta_X)$. When $x \in D(X, \delta_X)$, $x$ is *dead*. Otherwise, it is *live*.

Skip-free GKAT expressions are states in a skip-free automaton, so Definition 3.4.1 applies to them.

**Lemma 3.4.2.** *Let $(X, \delta_X)$ be a skip-free automaton and $x \in X$. Then $\mathcal{L}(x) = \emptyset$ if and only if $x \in D(X, \delta_X)$.*

Whether $e$ is dead can be determined by a simple depth-first search, since $e$ can reach only finitely many expressions in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ (Lemma 3.1.14). Furthermore, if a skip-free expression is dead, then it is provably 0.

**Lemma 3.4.3.** *Let $e \in GExp_{\mathsf{sf}}$. If $e$ is dead, then $e \equiv 0$.*

*Proof.* Recall the use of the notation $be = e +_b 0$. It suffices to prove that if $c \in BExp$ and $c\mathcal{L}(e) = \emptyset$, then $ce = e +_c 0 \equiv 0$. After all, if this is true and $e$ is dead then $1\mathcal{L}(e) = \mathcal{L}(e) = \emptyset$, and hence $e \equiv 1e \equiv 0$.

We proceed by induction on $e$. In the base case, there are two subcases:

- If $e = p \in \Sigma$, then for all $\alpha \leq c$ it holds that $\alpha p \in c\mathcal{L}(e)$. We thus find that $c =_{\mathsf{BA}} 0$, and therefore $p +_c 0 \equiv_0 p +_0 0 \equiv_0 0$ by (G2,G0).
- If $e = 0$, then $0 +_c 0 \equiv 0$ by (G1).

For the inductive step, there are three more cases.

- If $e = e_1 +_b e_2$ then $(c \wedge b)\mathcal{L}(e_1) = \emptyset$ and $(c \wedge \overline{b})\mathcal{L}(e_2) = \emptyset$. By induction, we have $(c \wedge b)e_1 \equiv 0$ and $(c \wedge \overline{b})e_2 \equiv 0$. We then derive

$$\begin{aligned} c(e_1 +_b e_2) &\equiv ce_1 +_b ce_2 && \text{(BD)} \\ &\equiv bce_1 +_b \overline{b}ce_2 && \text{(G9)} \\ &\equiv (c \wedge b)e_1 +_b (c \wedge \overline{b})e_2 && \text{(S6')} \\ &\equiv 0 +_b 0 && \text{(ind. hyp.)} \\ &\equiv 0 && \text{(G1)} \end{aligned}$$

- If $e = e_1 e_2$, then $c\mathcal{L}(e_1) = \emptyset$ or $\mathcal{L}(e_2) = \emptyset$. In the former case $e_1 +_c 0 \equiv 0$, so $c(e_1 e_2) \overset{\text{(G7)}}{\equiv} (ce_1)e_2 \equiv 0e_2 \overset{\text{(G6)}}{\equiv} 0$. In the latter case, $e_2 \equiv 0$, and thus $c(e_1 e_2) \equiv c(e_1 0) \overset{\text{(R0)}}{\equiv} c0 \overset{\text{(G1)}}{\equiv} 0$.
- If $e = e_1{}^{(b)} e_2$, then $\overline{b}\mathcal{L}(e_2) = \emptyset$, and so $\overline{b}e_2 \equiv 0$. By Proposition 3.2.15:

$$\begin{aligned} c(e_1{}^{(b)} e_2) &\equiv c(e_1{}^{(b)} \overline{b} e_2) && \text{(FP2')} \\ &\equiv c(e_1{}^{(b)} 0) \\ &\equiv c(e_1{}^{(b)} (00)) && \text{(R0)} \\ &\equiv c0 && \text{(G7',R0)} \\ &\equiv 0 && \text{(G1)} \end{aligned}$$

□

We are now ready to define $\lfloor - \rfloor$, the transformation on expressions promised above. Intuitively, we prune the dead subexpressions of $e$ by recursive descent: Whenever we find a subexpression that inevitably leads to failure, we set it to 0.

**Definition 3.4.4.** The map $\lfloor - \rfloor : \mathsf{sfGKAT} \to \mathsf{sfGKAT}$ is defined by

$$\lfloor 0 \rfloor = 0 \qquad \lfloor p \rfloor = p \qquad \lfloor e_1 +_b e_2 \rfloor = \lfloor e_1 \rfloor +_b \lfloor e_2 \rfloor$$

$$\lfloor e_1 e_2 \rfloor = \begin{cases} 0 & e_2 \text{ is dead} \\ \lfloor e_1 \rfloor \cdot \lfloor e_2 \rfloor & \text{otherwise} \end{cases} \qquad \left\lfloor e_1 {}^{(b)} e_2 \right\rfloor = \begin{cases} 0 & 0 +_b e_2 \text{ is dead} \\ \lfloor e_1 \rfloor {}^{(b)} \lfloor e_2 \rfloor & \text{otherwise} \end{cases}$$

The transformation above yields a term that is $\equiv$-equivalent to $e$, provided that we include the early termination axiom $e0 \equiv 0$. The proof is a simple induction on $e$, using Lemma 3.4.3.

**Lemma 3.4.5.** *For any $e \in GExp_{\mathsf{sf}}$, $e \equiv \lfloor e \rfloor$.*

*Proof.* We proceed by induction on $e$. In the base cases, the claim holds immediately, whether $e = 0$ or $e = p \in \Sigma$. For the inductive step, there are three cases.

- If $e = e_1 +_b e_2$, then $e_1 +_b e_2 \equiv \lfloor e_1 \rfloor +_b \lfloor e_2 \rfloor = \lfloor e_1 +_b e_2 \rfloor$.
- Suppose $e = e_1 e_2$. If $e_2$ is dead, then $e_2 \equiv 0$ by Lemma 3.4.3, and so $e_1 e_2 \equiv e_1 0 \overset{(\mathrm{R0})}{\equiv} 0 \equiv \lfloor e_1 e_2 \rfloor$ because $e_1 e_2$ is dead. Otherwise, $e_1 e_2 \equiv \lfloor e_1 \rfloor \cdot \lfloor e_2 \rfloor = \lfloor e_1 e_2 \rfloor$.
- Suppose $e = e_1 {}^{(b)} e_2$. If $\overline{b} e_2$ is dead, then $\overline{b} e_2 \equiv 0$ by Lemma 3.4.3, and so $e_1 {}^{(b)} e_2 \equiv e_1 {}^{(b)} \overline{b} e_2 \equiv e_1 {}^{(b)} 0 \overset{(\mathrm{G7',R0})}{\equiv} 0 = \left\lfloor e_1 {}^{(b)} e_2 \right\rfloor$ as seen above. Otherwise, we derive that $e_1 {}^{(b)} e_2 \equiv \lfloor e_1 \rfloor {}^{(b)} \lfloor e_2 \rfloor \equiv \left\lfloor e_1 {}^{(b)} e_2 \right\rfloor$. □

It remains to show that if $\mathcal{L}(e) = \mathcal{L}(f)$, then $\lfloor e \rfloor$ and $\lfloor f \rfloor$ are bisimilar. To this end, we need to relate the language semantics of $e$ and $f$ to their behaviour. As a first step, we note that behaviour that never leads to acceptance can be pruned from a skip-free automaton by removing transitions into dead states.

**Definition 3.4.6.** Let $(X, \delta_X)$ be a skip-free automaton. Define $\lfloor \delta_X \rfloor : X \to HX$ by

$$\lfloor \delta_X \rfloor (x)(\alpha) = \begin{cases} \bot & \delta_X(x)(\alpha) = (p, x'),\ x' \text{ is dead} \\ \delta_X(x)(\alpha) & \text{otherwise} \end{cases}$$

Moreover, language equivalence of two states in a skip-free automaton implies bisimilarity of those states, but only in the pruned version of that skip-free automaton. The proof works by showing that the relation on $X$ that connects states with the same language is, in fact, a bisimulation in $(X, \lfloor \delta_X \rfloor)$.

**Lemma 3.4.7.** *Let $(X, \delta_X)$ be a skip-free automaton and let $x, y \in X$. If $\mathcal{L}(x) = \mathcal{L}(y)$, then $x \leftrightarrows y$ in $(X, \lfloor \delta_X \rfloor)$.*

*Proof.* It suffices to prove that $R = \{(x,y) \mid \mathcal{L}(x) = \mathcal{L}(y)\}$ is a bisimulation on $(X, \lfloor \delta_X \rfloor)$. To see this, suppose that $\mathcal{L}(x) = \mathcal{L}(y)$. There are three cases to consider.

- If $\lfloor \delta_X \rfloor(x)(\alpha) = \bot$, then $\delta_X(x)(\alpha) = \bot$ or $\delta_X(x)(\alpha) = (p, x')$ with $x' \in D(X, \delta_X)$. Therefore, there is no word in $\mathcal{L}(x)$ of the form $\alpha w$, by Lemma 3.4.2. Thus, $\delta_X(y)(\alpha) = \bot$ or $\delta_X(y)(\alpha) = (p, y')$ where $\mathcal{L}(y') = \emptyset$, whence $y' \in D(X, \delta_X)$ by Lemma 3.4.2. In either case, $\lfloor \delta_X \rfloor(y)(\alpha) = \bot$.
- If $\lfloor \delta_X \rfloor(x)(\alpha) = (p, \checkmark)$ for some $p \in \Sigma$, then $\alpha p \in \mathcal{L}(x)$, and therefore $\alpha p \in \mathcal{L}(y)$. But then $\lfloor \delta_X \rfloor(y)(\alpha) = (p, \checkmark)$ as well.
- If $\lfloor \delta_X \rfloor(x)(\alpha) = (p, x') \in \Sigma \times X$, then $x'$ is not dead in $(X, \delta_X)$ by Lemma 3.4.2, and thus there exists some $w \in \mathcal{L}(x')$ s.t. $\alpha p w \in \mathcal{L}(x) = \mathcal{L}(y)$. We then also have $\delta_X(y)(\alpha) = (p, y')$ for some $y' \in X$ such that $w \in \mathcal{L}(y')$. $\mathcal{L}(y') \neq \emptyset$ means that $y'$ is live, so $\lfloor \delta_X \rfloor(y)(\alpha) = (p, y')$. An inductive argument on words shows that $\mathcal{L}(x') = \mathcal{L}(y')$, so $x' \, R \, y'$. □

There is one thing left to observe before we are ready to prove the second completeness theorem of the chapter. Whether we prune dead subexpressions from a skip-free expression (syntactic pruning) or we prune dead branches from the small-step semantics (semantic pruning), we end up assigning the same behaviour to skip-free expressions. In other words, $e \leftrightarrows f$ in $(GExp_{\mathsf{sf}}, \lfloor \delta_{\mathsf{sf}} \rfloor)$ if and only if $\lfloor e \rfloor \leftrightarrows \lfloor f \rfloor$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. An equivalent way to state this is as follows.

**Lemma 3.4.8.** *The pruning operator $\lfloor - \rfloor : (GExp_{\mathsf{sf}}, \lfloor \delta_{\mathsf{sf}} \rfloor) \to (GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ is an $H$-coalgebra homomorphism.*

*Proof.* We argue that $H(\lfloor - \rfloor) \circ \delta_{\mathsf{sf}}(e)(\alpha) = \lfloor \delta_{\mathsf{sf}} \rfloor \circ \lfloor - \rfloor(e)(\alpha)$ by induction on $e$, for any $\alpha \in At$. In the base case, $H(\lfloor - \rfloor) \circ \delta_{\mathsf{sf}}(0)(\alpha) = \bot = \lfloor \delta_{\mathsf{sf}} \rfloor(0)(\alpha) = \lfloor \delta_{\mathsf{sf}} \rfloor(\lfloor 0 \rfloor)(\alpha)$ and $H(\lfloor - \rfloor) \circ \delta_{\mathsf{sf}}(p)(\alpha) = (\alpha, \lfloor p \rfloor) = (\alpha, p) = \lfloor \delta_{\mathsf{sf}} \rfloor(p)(\alpha)$. For the inductive step, there are three cases.

- Suppose $e = e_1 +_b e_2$. If $\alpha \leq b$, then

$$H(\lfloor - \rfloor) \circ \delta_{\mathsf{sf}}(e_1 +_b e_2)(\alpha) = H(\lfloor - \rfloor) \circ \delta_{\mathsf{sf}}(e_1)(\alpha)$$

$$
\begin{aligned}
&= \lfloor \delta_{\mathsf{sf}} \rfloor (\lfloor e_1 \rfloor)(\alpha) && \text{(IH)} \\
&= \lfloor \delta_{\mathsf{sf}} \rfloor (\lfloor e_1 \rfloor +_b \lfloor e_2 \rfloor)(\alpha) \\
&= \lfloor \delta_{\mathsf{sf}} \rfloor (\lfloor e_1 +_b e_2 \rfloor)(\alpha)
\end{aligned}
$$

- Suppose $e = e_1 \cdot e_2$. If $e_2$ is dead, then so is $e_1 \cdot e_2$. Hence,

$$\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 \cdot e_2)(\alpha) = \bot = \delta_{\mathsf{sf}}(0)(\alpha) = \delta_{\mathsf{sf}}(\lfloor e_1 \cdot e_2 \rfloor)(\alpha)$$

  Otherwise, if $e_2$ is live, then we have three cases to consider.

  - If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 e_2)(\alpha) = \bot$, then we have two more subcases to consider.
    - If $\delta_{\mathsf{sf}}(e_1 e_2)(\alpha) = \bot$, then $\delta_{\mathsf{sf}}(e_1)(\alpha) = \bot$, meaning $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = \bot$. By induction, $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = \bot$, and so $\delta_{\mathsf{sf}}(\lfloor e_1 \cdot e_2 \rfloor)(\alpha) = \bot$.
    - If $\delta_{\mathsf{sf}}(e_1 e_2)(\alpha) = (p, e')$ with $e'$ dead, then we can exclude the case where $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, \checkmark)$, for then $e' = e_2$, which contradicts that $e_2$ is live. We then know that $e' = e_1' e_2$ such that $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e_1')$. Furthermore, since $e_2$ is live and $e'$ is dead, it must be the case that $e_1'$ is dead. We then find that $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = \bot$, and so $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = \bot$ by induction. We conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 e_2 \rfloor)(\alpha) = \delta_{\mathsf{sf}}(\lfloor e_1 \rfloor \lfloor e_2 \rfloor)(\alpha) = \bot$.
  - If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 e_2) = (p, \checkmark)$, then $\delta_{\mathsf{sf}}(e_1 e_2)(\alpha) = (p, \checkmark)$, which is impossible. We can therefore exclude this case.
  - If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 e_2) = (p, e')$, then $\delta_{\mathsf{sf}}(e_1 e_2)(\alpha) = (p, e')$ with $e'$ live. This gives us two more subcases.
    - If $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, \checkmark)$ and $e' = e_2$, then $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = (p, \checkmark)$, and so by induction $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = (p, \checkmark)$. Thus, $\delta_{\mathsf{sf}}(\lfloor e_1 e_2 \rfloor)(\alpha) = (p, \lfloor e' \rfloor)$.
    - If $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e_1')$ and $e' = e_1' e_2$, then $e_1'$ must be live. We then have $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = (p, e_1')$, and so $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = (p, \lfloor e_1' \rfloor)$ by induction. We conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 e_2 \rfloor)(\alpha) = (p, \lfloor e_1' \rfloor \lfloor e_2 \rfloor) = (p, \lfloor e \rfloor)$.

- Suppose $e = e_1 {}^{(b)} e_2$. If $\overline{b} e_2$ is dead, then so is $e_1 {}^{(b)} e_2$; hence

$$\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = \bot = \delta_{\mathsf{sf}}(0)(\alpha) = \delta_{\mathsf{sf}}(\left\lfloor e_1 {}^{(b)} e_2 \right\rfloor)(\alpha)$$

  Otherwise, if $\overline{b} e_2$ is live, then we first consider the case where $a \in \overline{b}$.

- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = \bot$, then $\delta_{\mathsf{sf}}(e_2)(\alpha) = \bot$ or $\delta_{\mathsf{sf}}(e_2)(\alpha) = (p, e_2')$ with $e_2'$ dead. In either case, $\lfloor \delta_{\mathsf{sf}} \rfloor (e_2)(\alpha) = \bot$, and so $\delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = \bot$ by induction. We conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 {}^{(b)} e_2 \rfloor)(\alpha) = \delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = \bot$.
- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = (p, \checkmark)$, then $\delta_{\mathsf{sf}}(e_2)(\alpha) = (p, \checkmark)$. Thus, $\lfloor \delta_{\mathsf{sf}} \rfloor (e_2)(\alpha) = (p, \checkmark)$ and so we have $\delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = (p, \checkmark)$ by induction. We then conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 {}^{(b)} e_2 \rfloor)(\alpha) = \delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = (p, \checkmark)$.
- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = (p, e_2')$, then $\delta_{\mathsf{sf}}(e_2)(\alpha) = (p, e_2')$ with $e_2'$ live. It then follows that $\lfloor \delta_{\mathsf{sf}} \rfloor (e_2)(\alpha) = (p, e_2')$, and so $\delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = (p, \lfloor e_2' \rfloor)$ by induction. We then conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 {}^{(b)} e_2 \rfloor)(\alpha) = \delta_{\mathsf{sf}}(\lfloor e_2 \rfloor)(\alpha) = (p, \lfloor e_2' \rfloor)$.

It remains to consider the case where $\alpha \leq b$.

- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = \bot$, then we have two more subcases to consider.
  - If $\delta_{\mathsf{sf}}(e_1 {}^{(b)} e_2)(\alpha) = \bot$, then $\delta_{\mathsf{sf}}(e_1)(\alpha) = \bot$ as well, since $\alpha \leq b$. This means that $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = \bot$, and so $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = \bot$ by induction. We conclude that $\delta_{\mathsf{sf}}(\lfloor e_1 {}^{(b)} e_2 \rfloor)(\alpha) = \bot$.
  - If $\delta_{\mathsf{sf}}(e_1 {}^{(b)} e_2)(\alpha) = (p, e')$ with $e'$ dead, then since $\alpha \leq b$ we must have that $e' = e_1' e_1 {}^{(b)} e_2$, with $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e_1')$. Since $\overline{b} e_2$ is live, so is $e_1 {}^{(b)} e_2$. It then follows that $e_1'$ is dead, and so $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = \bot$. By induction, $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = \bot$, and so $\delta_{\mathsf{sf}}(\lfloor e_1 {}^{(b)} e_2 \rfloor)(\alpha) = \bot$.
- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = (p, \checkmark)$, then $\delta_{\mathsf{sf}}(e_1 {}^{(b)} e_2)(\alpha) = (p, \checkmark)$, which is impossible when $\alpha \leq b$. We can therefore exclude this case.
- If $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1 {}^{(b)} e_2)(\alpha) = (p, e')$, then $\delta_{\mathsf{sf}}(e_1 {}^{(b)} e_2)(\alpha) = (p, e')$ with $e'$ live. Since $\alpha \leq b$, we have that $\delta_{\mathsf{sf}}(e_1)(\alpha) = (p, e_1')$ and $e' = e_1' \cdot e_1 {}^{(b)} e_2$. Since $e'$ and $e_1 {}^{(b)} e_2$ are live, so is $e_1'$. We then find that $\lfloor \delta_{\mathsf{sf}} \rfloor (e_1)(\alpha) = (p, e_1')$, meaning $\delta_{\mathsf{sf}}(\lfloor e_1 \rfloor)(\alpha) = (p, \lfloor e_1' \rfloor)$ by induction. We conclude by deriving

$$\delta_{\mathsf{sf}}(\left\lfloor e_1 {}^{(b)} e_2 \right\rfloor)(\alpha) = (p, \lfloor e_1' \rfloor \cdot \lfloor e_1 \rfloor {}^{(b)} \lfloor e_2 \rfloor) = (p, \lfloor e' \rfloor)$$

□

Recall that $e \equiv f$ denotes $\mathsf{sfGKAT} \vdash e = f$ (see Figure 3.8). In particular, there is no need for the uniqueness axiom in skip-free GKAT, so it is not included in sfGKAT. We now have all the ingredients necessary to prove Theorem 3.2.19.

**Theorem 3.2.19** (Completeness II)**.** *Let $e_1, e_2 \in GExp_{\mathsf{sf}}$. If $\mathcal{L}(e_1) = \mathcal{L}(e_2)$, then $e_1 \equiv e_2$.*

*Proof.* If $\mathcal{L}(e_1) = \mathcal{L}(e_2)$, then by Lemma 3.4.7, $e_1 \leftrightarrows e_2$ in $(GExp_{\mathsf{sf}}, \lfloor \delta_{\mathsf{sf}} \rfloor)$, which by Lemma 3.4.8 implies that $\lfloor e_1 \rfloor \leftrightarrows \lfloor e_2 \rfloor$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. From Theorem 3.2.18 we know that $\lfloor e_1 \rfloor \equiv_0 \lfloor e_2 \rfloor$, and therefore $e_1 \equiv e_2$ by Lemma 3.4.5. □

## 3.5 Relation to GKAT

So far, we have seen the technical development of skip-free GKAT without much reference to the original development of GKAT. In this section, we make the case that the semantics of skip-free GKAT is merely a simplified version of the semantics of GKAT, and that the two agree on which expressions are equivalent after seeing skip-free GKAT expressions as GKAT expressions. More precisely, we identify the bisimulation and language semantics of skip-free GKAT given in Section 3.2 with instances of the existing bisimulation and language semantics of GKAT proper. The main takeaway is that two skip-free GKAT expressions are equivalent in our semantics precisely when they are equivalent when interpreted as proper GKAT expressions in the existing semantics.

$$\frac{\alpha \leq b}{b \Rightarrow \alpha} \quad \frac{\alpha \leq b \quad e_1 \Rightarrow \alpha}{e_1 +_b e_2 \Rightarrow \alpha} \quad \frac{\alpha \leq \bar{b} \quad e_2 \Rightarrow \alpha}{e_1 +_b e_2 \Rightarrow \alpha} \quad \frac{\alpha \leq b \quad e_1 \xrightarrow{\alpha|p} e'}{e_1 +_b e_2 \xrightarrow{\alpha|p} e'} \quad \frac{\alpha \leq \bar{b} \quad e_2 \xrightarrow{\alpha|p} e'}{e_1 +_b e_2 \xrightarrow{\alpha|p} e'}$$

$$\frac{}{p \xrightarrow{\alpha|p} 1} \quad \frac{e_1 \Rightarrow \alpha \quad e_2 \Rightarrow \alpha}{e_1 e_2 \Rightarrow \alpha} \quad \frac{e_1 \Rightarrow \alpha \quad e_2 \xrightarrow{\alpha|p} e'}{e_1 e_2 \xrightarrow{\alpha|p} e'} \quad \frac{e_1 \xrightarrow{\alpha|p} e'}{e_1 e_2 \xrightarrow{\alpha|p} e' \fatsemi e_2}$$

$$\frac{\alpha \leq b \quad e \xrightarrow{\alpha|p} e'}{e^{(b)} \xrightarrow{\alpha|p} e' \fatsemi e^{(b)}} \quad \frac{\alpha \leq \bar{b}}{e^{(b)} \Rightarrow \alpha}$$

**Figure 3.9:** The transition function $\tilde{\delta} : GExp \to (\bot + \checkmark + \Sigma \times GExp)^{At}$ defined inductively. Here, $e_1 \fatsemi e_2$ is $e_2$ when $e = 1$ and $e_1 \cdot e_2$ otherwise, $b \in \mathsf{BExp}$, $p \in \Sigma$, and $e, e', e_i \in GExp$.

We begin with an intermediary step, a slightly different but equivalent small-step semantics for GKAT. We equip $GExp$ with the GKAT automaton structure $(GExp, \tilde{\delta})$ outlined in Figure 3.9. The definition of $\tilde{\delta}$ diverges slightly from the small-step semantics found in Figure 3.2. Fortunately, this does not make a difference in terms of the bisimulation semantics: Two expressions are bisimilar in $(GExp, \tilde{\delta})$ if and only if they are bisimilar in the original semantics.

There is a fairly easy way to convert a skip-free automaton into a GKAT automaton: Simply reroute all accepting transitions into a new state $\top$, that accepts

immediately, and leave the other transitions the same.

**Definition 3.5.1.** Given a skip-free automaton $(X, \delta_X)$, we define the automaton $\mathsf{embed}(X, \delta_X) = (X + \top, \tilde{\delta_X})$, where $\tilde{\delta_X}$ is defined by

$$\tilde{\delta_X}(x)(\alpha) = \begin{cases} \checkmark & x = \top \\ (p, \top) & \delta_X(x)(\alpha) = (p, \checkmark) \\ \delta_X(x)(\alpha) & \text{otherwise} \end{cases}$$

We can show that two states are bisimilar in a skip-free automaton if and only if these same states are bisimilar in the corresponding GKAT automaton.

**Lemma 3.5.2.** *Let $(X, \delta_X)$ be a skip-free automaton, and let $x, y \in X$. Then*

$$x \leftrightarrows y \text{ in } (X, \delta_X) \text{ if and only if } x \leftrightarrows y \text{ in } \mathsf{embed}(X, \delta_X)$$

The syntactic skip-free automaton $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ can of course be converted to a GKAT automaton in this way. It turns out that there is a very natural way of correlating this automaton to the syntactic GKAT automaton $(GExp, \tilde{\delta})$.

**Lemma 3.5.3.** *The relation $\{(e, e) \mid e \in GExp_{\mathsf{sf}}\} \cup \{(\top, 1)\}$ is a bisimulation between $\mathsf{embed}(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ and $(GExp, \tilde{\delta})$.*

We now have everything to relate the bisimulation semantics of skip-free GKAT expressions to the original bisimulation semantics of GKAT expressions.

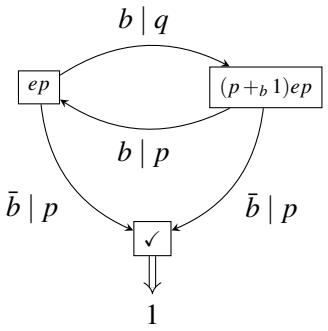
**Lemma 3.5.4.** *Let $e, f \in GExp_{\mathsf{sf}}$. The following holds:*

$$e \leftrightarrows f \text{ in } (GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}}) \text{ if and only if } e \leftrightarrows f \text{ in } (GExp, \tilde{\delta})$$

*Proof.* We derive using Lemmas 3.5.2 and 3.5.3, as follows: since the graph of embed is a bisimulation, $e \leftrightarrows f$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ if and only if $e \leftrightarrows f$ in $\mathsf{embed}(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$ if and only if $e \leftrightarrows f$ in $(GExp, \tilde{\delta})$. In the last step, we use the fact that bisimulations are composable, as in Lemma 2.2.12. □

*Example* 3.5.5. We can now make a previous statement precise. Namely, that there are GKAT programs of the form $ep$ with $e \in GExp$ and $p \in \Sigma$ such that $ep$ is not expressible

in skip-free GKAT. More precisely, $ep$ is not bisimilar to any skip-free $g \in GExp_{\mathsf{sf}}$, where bisimilarity is considered between $(GExp, \tilde{\delta})$ (equivalently, $(GExp, \delta)$) and $\mathsf{embed}(GExp_{\mathsf{sf}}, \delta)$. The example presented before was $(q(p +_b 1))^{(b)} p$ for $0 < b < 1$ and $p \neq q \in \Sigma$, whose corresponding automaton is




$$e = (q(p +_b 1))^{(b)} p$$

Observe that $\neg(ep \leftrightarrows (p +_b 1)ep)$, so that the automaton above has no proper quotients. Also note that the automaton above clearly has an underlying skip-free automaton. If there were a $g \in GExp_{\mathsf{sf}}$ such that $ep \leftrightarrows g$, then applying $\mathsf{grph}_*$ to the underlying skip-free automaton would produce a bisimulation collapse of a well-layered prechart, which we know must also be well-layered by Definition 2.4.2 and Theorem 2.4.12. The prechart corresponding to the automaton above is not well-layered.

**Language semantics**

Since skip-free GKAT expressions are also GKAT expressions, we are left with two language interpretations of skip-free expressions. Let us use $\widehat{\mathcal{L}}$ to denote the GKAT language semantics from Definition 3.1.6. Then $\widehat{\mathcal{L}}$ can be expressed in terms of $\mathcal{L}$.

**Lemma 3.5.6.** *For $e \in GExp_{\mathsf{sf}}$, it holds that $\widehat{\mathcal{L}}(e) = \mathcal{L}(e) \cdot At$.*

As an easy consequence of the above, we find that the two semantics must identify the same skip-free GKAT expressions.

**Lemma 3.5.7.** *For $e, f \in GExp_{\mathsf{sf}}$, we have $\mathcal{L}(e) = \mathcal{L}(f)$ if and only if $\widehat{\mathcal{L}}(e) = \widehat{\mathcal{L}}(f)$.*

By Theorem 3.2.19, these properties imply that sfGKAT also axiomatizes relational equivalence of skip-free GKAT-expressions. Recall that a relational interpretation of GKAT is a triple $\sigma = (S, \mathsf{eval}, \mathsf{sat})$, where $S$ is a set, $\mathsf{eval} \colon \Sigma \to \mathcal{P}(S \times S)$, and $\mathsf{sat} \colon At \to \mathcal{P}(S)$, and generates a function $[\![-]\!]_\sigma : GExp \to \mathcal{P}(S \times S)$. We obtain the notion of relational semantics for skip-free GKAT by interpreting skip-free GKAT expressions as GKAT expressions.

**Corollary 3.5.8.** *Let $e, f \in GExp_{\mathsf{sf}}$, we have $e \equiv f$ if and only if $[\![e]\!]_\sigma = [\![f]\!]_\sigma$ for all relational interpretations $\sigma$.*

**Provable Equivalence**

Finally, we relate provable equivalences between skip-free GKAT expressions to those provable between proper GKAT expressions, showing that proofs of equivalence for skip-free GKAT expressions can be replayed in the larger calculus, without UA.

The axioms of GKAT are provided in Figure 3.5. Since skip-free GKAT expressions are also GKAT expressions, we are left with four notions of provable equivalence for GKAT expressions: As skip-free expressions or GKAT expressions in general, either with or without (R0). These are related as follows.

**Theorem 3.5.9.** *Let $e, f \in GExp_{\mathsf{sf}}$. Then*

*(1)* $\mathsf{sfGKAT}_0 \vdash e = f$ *if and only if* $\mathsf{GKAT}_0 \vdash e = f$*, and*

*(2)* $\mathsf{sfGKAT} \vdash e = f$ *if and only if* $\mathsf{GKAT} \vdash e = f$.

*Proof.* Left-to-right is straightforward because every axiom of sfGKAT is an axiom of GKAT. To go from right-to-left in either (1) or (2), it really needs to be shown that the axioms (G4) and (G5) can safely be avoided. This can be seen from the completeness theorems for skip-free GKAT.

For (1), suppose $\mathsf{GKAT}_0 \vdash e = f$. Then $e \leftrightarrows f$ in $(GExp, \delta)$. As we have seen, this means that $e \leftrightarrows f$ in $(GExp, \tilde{\delta})$, and as skip-free expressions, $e \leftrightarrows f$ in $(GExp_{\mathsf{sf}}, \delta_{\mathsf{sf}})$. By the completeness of $\mathsf{sfGKAT}_0$, $\mathsf{sfGKAT}_0 \vdash e = f$.

For (2), suppose that $\mathsf{GKAT} \vdash e = f$. Then by soundness, $\widehat{\mathcal{L}}(e) = \widehat{L}(f)$. We therefore have $\mathcal{L}(e) \cdot At = \mathcal{L}(f) \cdot At$, which implies $\mathcal{L}(e) = \mathcal{L}(f)$. By the completeness of sfGKAT, $\mathsf{sfGKAT} \vdash e = f$. □

Note that the uniqueness axiom does not appear in the above theorem. One consequence of this is that our completeness theorems for skip-free GKAT are genuinely partial completeness results for GKAT: They tell us that the axiomatization of GKAT is complete for skip-free GKAT expressions.

## 3.6 Related Work

This chapter fits into a larger research program focused on understanding the logical and algebraic content of uninterpreted programs. Kleene's paper introducing the

algebra of regular languages [Kle56] was a foundational contribution to this research program, containing an algebraic account of mechanical programming and some of its sound equational laws. Kleene's paper also contains an interesting completeness problem: Give a complete description of the equations satisfied by the algebra of regular languages. Salomaa was the first to provide a sound and complete axiomatization of language equivalence for regular expressions [Sal66].

The axiomatization in op. cit. included an inference rule with a side condition that prevented it from being *algebraic* in the sense that the validity of an equation is not preserved when substituting letters for arbitrary regular expressions. Nevertheless, this inspired axiomatizations of several variations and extensions of Kleene algebra [Wag+19; Smo+20; Sch+22], as well as Milner's axiomatization of the algebra of star behaviours [Mil84], and other related process algebras [BW90]. The side condition introduced by Salomaa is often called the *empty word property*, an early version of a concept from process theory called *guardedness*[9] that is also fundamental to the theory of iteration [BÉ93].

Our axiomatization of skip-free GKAT *is* algebraic due to the lack of a guardedness side-condition (it is an equational *Horn theory* [Mak87]). This is particularly desirable because it allows for an abundance of other models of the axioms, including relational models like in Example 3.1.5. Kozen proposed an algebraic axiomatization of Kleene algebra that is sound and complete for language equivalence [Koz91], which has become the basis for a number of axiomatizations of other Kleene algebra variants [Fos+15; Kap+19; Kap+20; Wag+20] including Kleene algebra with tests [KS96]. KAT also has a plethora of relational models, which are desirable for reasons we hinted at in the beginning of the thesis.

GKAT is a fragment of KAT that was first identified in [KT08]. It was later given a sound and complete axiomatization in [Smo+20], although the axiomatization is neither algebraic nor finite (it includes UA, an axiom scheme that stands for infinitely many axioms that all have a guardedness side condition).

Despite the existence of an algebraic axiomatization of language equivalence in KAT, GKAT has resisted algebraic axiomatization so far. Skip-free GKAT happens to

[9] This is a different use of the word "guarded" than in "guarded Kleene algebra with tests". In the context of process theory, a recursive specification is guarded if every recursive call occurs within the scope of an action.

be a fragment of GKAT in which every expression is guarded, thus eliminating the need for the side condition in Figure 3.5 and allowing for an algebraic axiomatization. One approach to an algebraic axiomatization of full GKAT involves the introduction of an order. For instance, an inequational axiomatization resembling that of KAT can be gleaned from my recent preprint [Sch22a]. Unfortunately, one of my axioms in op. cit. implicitly includes a Salomaa-like side condition, and is not algebraic. An alternative approach to an algebraic axiomatization takes inspiration from process algebra: The GKAT axioms for bisimilarity of ground terms (without recursion) can be obtained from the small-step semantics of GKAT using a GSOS law [Ace94; Ace+11a; Ace+11b]. Following a common approach to axiomatization in process algebra, we could extend the base terms with iteration. The problem with this approach is that it produces a calculus that is more expressive than GKAT (it is equivalent to a calculus we will see in Chapter 4).

On the other hand, our algebraic axiomatization of skip-free GKAT was inspired by (and relies on) a completeness theorem in process algebra: the algebraic axiomatization of one-free star behaviours due to Grabmayer and Fokkink [GF20]. In op. cit., Grabmayer and Fokkink proved that Milner's axioms for star behaviours [Mil84] are complete for bisimilarity of one-free star expressions. This gave a partial solution to Milner's completeness problem, which was recently solved in full by Grabmayer [Gra22]. Our proof that bisimulation GKAT is a complete axiomatization of bisimilarity for skip-free GKAT expressions is based on the observation that skip-free bisimulation GKAT is equivalent to the deterministic fragment of Grabmayer and Fokkink's one-free regular expressions modulo bisimilarity. This observation allowed us to reduce the completeness problem for skip-free GKAT to Grabmayer and Fokkink's work, bypassing the intricate combinatorics in their completeness proof.

The idea of reducing one completeness problem to another is common in Kleene algebra. For instance, it is behind the completeness proof of KAT [KS96]. Cohen also reduced *weak Kleene algebra* as an axiomatization of star expressions modulo simulation to *monodic trees* [Coh09], whose completeness was conjectured by Takai and Furusawa [TF06]. Grabmayer's solution to the completeness problem of regular expressions modulo bisimilarity [Gra22] can also be seen as a reduction to the one-free case, since his *crystallization* procedure produces an automaton that can be solved

using the technique found in [GF20]. Other instances of reductions include [Coh94; And+14; Dou+19; Wag+20; Kap+19; Kap+18; LS17; PW22; KM14]. Recent work has started to study reductions and their compositionality properties [Dou+19; Kap+20; PRW21].

## 3.7 Discussion

This chapter continues the study of an efficient fragment of Kleene Algebra with Tests (KAT) initiated in [Smo+20], where the authors introduce GKAT and provide an efficient decision procedure for equivalence. The authors of [Smo+20] proposed a candidate axiomatization, but left open two questions.

1. The first question concerned the existence of an algebraic axiomatization. This is essential to enable compositional analysis.
2. The second question left open in [Smo+20] was whether an axiomatization that did not require an axiom scheme was possible.

In this chapter, a large fragment of GKAT is identified, which we call *skip-free* GKAT (sfGKAT), that can be axiomatized algebraically without relying on an axiom scheme. We show that the axiomatization works well for both bisimilarity and language equivalence by proving completeness results for both semantics. Having the two semantics is interesting from a verification point of view as it gives access to different levels of precision when analyzing program behaviour. In the case of GKAT, we also saw that the completeness problem for one semantics could be reduced to the other.

We provide a reduction of the completeness proof for language semantics to the one for bisimilarity. Our approach enables two things: It breaks down the completeness proofs and reuses some techniques while also highlighting the exact difference between the two equivalences (captured by the axiom $e \cdot 0 \equiv 0$ which does not hold for bisimilarity). We also showed that proofs of equivalence in skip-free GKAT transfer without any loss to proofs of equivalence in GKAT.

There are several directions for future work. The bridge between process algebra and Kleene algebra has not been exploited to its full potential. The fact that we could reuse results by Grabmayer and Fokkink [Gra22; GF20] was a major step towards completeness. An independent proof would have been much more complex and very likely required the development of technical tools resembling those in [Gra22; GF20].

We hope the results in this chapter can be taken further and more results can be exchanged between the two communities to solve open problems.

The completeness problem for full GKAT remains open, but our completeness results for skip-free GKAT are encouraging. I believe they show a path towards studying whether an algebraic axiomatization can be devised or a negative result can be proved. A first step in exploring a completeness result would be to try extending Grabmayer's completeness result [Gra22] to a setting with output variables—this is a non-trivial exploration, but we are hopeful will yield new tools for completeness (see Section 4.5). As mentioned in the introduction, NetKAT [And+14] (and its probabilistic variants [Fos+16; Smo+17]) have been one of the most successful extensions of KAT. We believe the step from skip-free GKAT to a skip-free guarded version of NetKAT is also a worthwhile exploration.

## Chapter 4

# Effectful Process Calculi

Process algebras have a long tradition, notably in the study of concurrency, pioneered by seminal works of Milner [Mil84; Mil80] and many others [Bae05; BW90; BBR09]. Conceptually, a process algebra is a structure consisting of a set of processes (states in a running, state-based, interactive machine), operations for composing and combining processes, and equations for reasoning about process equivalence. In Milner's CCS [Mil80], where processes branch nondeterministically, there is a binary operation that constructs from a pair of processes $e$ and $f$ the process $e+f$ that nondeterministically chooses between executing either $e$ or $f$. This acts precisely like the join operation in a semilattice. In fact, elements of a free semilattice are exactly sets, as the free semilattice generated by a set $X$ is the set $\mathcal{P}^+_\omega X$ of finite nonempty subsets of $X$ [Man76]. This is our first example of a more general phenomenon: The types of branching found in process algebras are often captured by a free algebra construction[1]. We refer to branching captured by free algebra constructions as *effectful*, in reference to Plotkin and Power's *algebraic effects* [PP02][2].

For another example of effectful branching, in the probabilistic process algebra literature, the process denoted $e \oplus_p f$ flips a weighted coin and runs $e$ with probability $p$ and $f$ with probability $1-p$. The properties of $\oplus_p$ are axiomatized and studied in convex algebra, an often revisited algebraic theory of probability [Sto49; Świ74; PR95; And99; BSV21]. The free convex algebra on a set $X$ is the set $\mathcal{D}_\omega X$ of finitely supported probability distributions on $X$ [Sto49; BSV19; Jac10].

A third example is GKAT, where the process $e +_b f$ proceeds with $e$ if a certain

[1] I.e., a finitary monad [Law63].

[2] The words *effect* and *branching* usually refer to *strong* and *commutative* monads respectively (see for example [PP02; Cîr17]). For the sake of simple terminology, we use these terms more broadly in this chapter. See the text under *Algebraic effects* in Related Work (Section 4.6) for more detail.

Boolean predicate $b$ holds and otherwise proceeds with $f$, emulating the if-then-else constructs of imperative programming languages [Man91; BT83; McC61]. If the predicates are taken from a finite Boolean algebra $2^{At}$, the free algebra of if-then-else clauses on a set $X$ is the function space $X^{At}$. This explains why models of GKAT programs often take the form of functions $At \to X$.

**Effectful process algebra**

This chapter proposes a framework, *effectful process algebra*, in which these languages can be uniformly described and studied. Roughly, an effectful process algebra is an algebraic structure consisting of processes whose branching behaviour can be captured by an algebraic theory. The *algebra of regular behaviours* (or ARB) introduced by Milner [Mil84] is a prototypical example: ARB employs nondeterministic choice as a branching operation, prefixing of terms by atomic actions, a constant representing deadlock, variables, and a recursion operator for each variable. Specifications are interpreted operationally in the style of structural operational semantics [Plo04], which sees the set $Exp$ of all process terms as one large labelled transition system. This is captured succinctly as a coalgebra (Definition 2.2.6) of the form

$$\beta : Exp \to \mathcal{P}_\omega(Var + Act \times Exp) \tag{4.1}$$

where $\mathcal{P}_\omega$ is the finite powerset functor (only finitely branching processes can be specified in ARB). From a technical point of view, $\mathcal{P}_\omega$ is the functor part of the monad on the category of sets presented by the algebraic theory of semilattices with bottom.

By substituting the finite powerset functor in (4.1) with other monads presented by algebraic theories, we obtain a parametrized family of process types that covers the examples above and a general framework for studying the processes of each type. Instantiating the framework with a given algebraic theory gives a fully expressive specification language for processes with branching given by that theory and a complete axiomatization of behavioural equivalence for specifications.

**Effectful process calculi**

I begin the chapter by introducing a family of process calculi, called *effectful process calculi*, that is parametrized by a kind of equational algebraic theory. I give each effectful process calculus an operational and a denotational semantics and show

that they coincide. I also show that effectful process calculi admit a *Kleene theorem*, which says that every finitary behaviour can be specified by an expression in the syntax of the calculus. This naturally leads to a sound and complete axiomatization of equivalence.

The specific kinds of algebraic theories covered by the framework are finitary and have a notion of *unguarded recursion* built-in. For example, given any term $p(x,\vec{y})$ in the language of semilattices with bottom, there is a canonical term $q$ such that $q = p(q,\vec{y})$ up to the axioms—namely, $q = p(0,\vec{y})$ ($q$ is the least fixed-point of $x \mapsto p(x,\vec{y})$). In ARB, this allows recursion to be performed on arbitrary terms, even those in which variable outputs appear in the initial stages of execution. Algebraic theories included in the effectful process algebra framework are equipped with a fixed-point operator that plays a role analogous to $p(u,\vec{y}) \mapsto p(0,\vec{y})$ from ARB.

**Star Fragments**

One striking feature of many of the specification languages we construct is that they contain a fragment consisting of nonstandard analogues of regular expressions. We call these expressions *star expressions* and the fragment composed of star expressions the *star fragment*.

Star fragments extend several existing analogues of regular expressions found in the process theory literature, including basic process algebra [BK88] and Andova's probabilistic basic process algebra [And99], by adding recursion operators modelled after the Kleene star. Milner is the first to notice the star fragment of ARB in [Mil84], whose syntax is given by regular expressions. He observes that the algebra of processes denoted by star expressions is more unruly than Kleene's algebra of regular languages, and that it is not clear what the appropriate axiomatization should be. He offers a reasonable candidate based on Salomaa's first axiomatization of Kleene algebra [Sal66], but ultimately leaves completeness as an open problem. This problem was subject to many years of extensive research [FZ94; Fok97; FZ97; BCG06; GF20; Gra21] before Grabmayer announced his positive solution [Gra22].

Replacing nondeterministic choice with the `if-then-else` branching structure of GKAT (and an analogous unguarded iteration operator), we obtain the process behaviours explored in *bisimulation* GKAT, the process algebra inspired take on GKAT from Chapter 3. This makes the open problem of axiomatizing GKAT (without

the use of an extremely powerful axiom scheme like the *uniqueness axiom* (UA) from Chapter 3 [Smo+20]), yet another problem of axiomatizing a star fragment.

Our general characterization of star fragments puts all these languages under one umbrella, and shows how they are derived canonically from a single abstract framework. In this chapter, I give each star fragment both an operational and a denotational semantics, show that the operational and denotational semantics coincide with the semantics of their ambient effectful process calculi, and propose a uniform axiomatization of these algebras by generalizing Milner's axioms for regular expressions modulo bisimilarity (based on Salomaa's [Sal66]). Unlike effectful process calculi, star fragments do not admit a Kleene theorem. As we saw in Chapter 2, the lack of a Kleene theorem is a serious barrier to finding a completeness proof for this kind of axiomatization. At the end of the chapter, an analogue of UA from Chapter 3 is used to obtain a completeness theorem for star fragments.

**Outline**

In summary, the contributions of this chapter are as follows:

- I present a family of process types parametrized by what are called *branching theories* (Section 4.1), algebraic theories equipped with a basic notion of recursion (this allows us to resolve unguarded recursive calls). For each process type, I give a uniform syntax and operational semantics (Section 4.2). I show how these can be instantiated to concrete algebraic theories including guarded algebras and pointed convex algebras (and more). These provide, respectively, a calculus of processes capturing control flow of simple imperative programs and a calculus of probabilistic processes.
- I define an associated denotational semantics and show that it agrees with the operational semantics (Section 4.3). This coincidence result is important in order to prove completeness of the uniform axiomatization we propose for each process type (Section 4.4).
- Finally, I define the star fragments of effectful process algebras and propose a sound axiomatization of behavioural equivalence of star expressions (Section 4.5). I also show that star fragments of concrete instances of our calculi yield known examples in the literature like, GKAT [Smo+20; Sch+21]. The chapter ends with a general completeness theorem for star fragments that uses

an analog of the uniqueness axiom from Chapter 3.

Related work is discussed in more detail in Section 4.6.

## 4.1 A Parametrized Family of Process Types

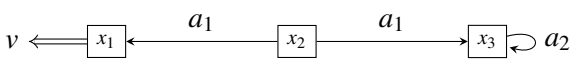
In this section, I present a family of process types parametrized by a *branching theory*, an algebraic theory equipped with a kind of recursion. The processes of interest are stateful, meaning they consist of a set of states and a suitably structured set of transitions between states. Stateful systems fit neatly into the general framework of *coalgebra* [Rut00], which stipulates that the type of structure carried by the transitions can be encoded in an endofunctor on the category **Set** of sets and functions. We covered the basics of coalgebra in Chapter 2.

In this chapter, we consider coalgebras for endofunctors of the form

$$B_M = M(Var + Act \times \mathrm{Id}) \tag{4.2}$$

for fixed sets $Var$ and $Act$ and a specific kind of functor $M : \mathbf{Set} \to \mathbf{Set}$. Here, Id denotes the identity functor on **Set**. Intuitively, there are two layers to the behaviours of these coalgebras. The first layer consists of either an *output variable* in $Var$ or an *action* from $Act$ that moves on to another state, and the other layer (encoded by $M$) combines output variables and action steps in a structured way.

*Example* 4.1.1. When $M = \mathcal{P}_\omega$, we obtain Milner's nondeterministic processes [Mil84]. Coalgebras for $B_{\mathcal{P}_\omega}$ are functions of the form $\beta : X \to \mathcal{P}_\omega(Var + Act \times X)$, or labelled transition systems with an additional decoration by variables. Write $x \xrightarrow{a} y$ to mean $(a, y) \in \beta(x)$ and $x \Rightarrow v$ to mean $v \in \beta(x)$. The image below posits a well-defined $B_{\mathcal{P}_\omega}$-coalgebra $\beta : \{x_1, x_2, x_2\} \to \mathcal{P}_\omega(Var + Act \times \{x_1, x_2, x_3\})$.




$$\beta(x_1) = \{v\}$$
$$\beta(x_2) = \{(a_1, x_1), (a_1, x_3)\}$$
$$\beta(x_3) = \{(a_2, x_3)\}$$

Here, $a_1, a_2 \in Act$ and $v \in Var$.

*Remark* 4.1.2. There are many kinds of nondeterminism studied in the program semantics literature [BW81]. Milner's processes specifically exhibit *angelic* nondeter-

minism, a form of nondeterministic choice that favours unproblematic branches of execution (as seen from the axiom $r+0=r$).

**Algebraic Theories and Their Monads**

We are particularly interested in $B_M$-coalgebras when $M$ is the functor component of a free-algebra construction that captures the desired type of branching. This captures the case where $M=\mathcal{P}_\omega$ (Example 4.1.1), but it also captures many other examples (see the end of this subsection).

**Definition 4.1.3.** A *monad* is a triple $(M,\eta,\mu)$ consisting of an endofunctor $M$ on **Set** and a pair of natural transformations $\eta\colon \mathrm{Id}\Rightarrow M$ and $\mu\colon MM\Rightarrow M$, called the *unit* and *multiplication* respectively, satisfying

$$\mu\circ\eta_M=\mathrm{id}_M=\mu\circ M(\eta)\qquad \mu_M\circ\mu=M(\mu)\circ\mu$$

**Definition 4.1.4.** A *finitary equational theory* is a pair $(S,\mathsf{EQ})$ consisting of a sequence of sets $S=\{S_n\}_{n\in\mathbb{N}}$ called an *algebraic signature* and a set $\mathsf{EQ}\subseteq(S^*\mathbb{N})\times(S^*\mathbb{N})$ of formal equations between *S-terms*, where given a set $X$ the set $S^*X$ of $S$-terms is defined

$$S^*X\ni t_i ::= x\in X \mid \sigma(t_1,\ldots,t_n) \qquad (\sigma\in S_n)$$

An *S-algebra* is a pair $(X,\alpha)$ consisting of a set $X$ and for each $n\in\mathbb{N}$ and $\sigma\in S_n$ a function $\sigma^\alpha\colon X^n\to X$. An *S-algebra homomorphism* $h\colon (X,\alpha_X)\to(Y,\alpha_Y)$ is a function $h\colon X\to Y$ such that for any $n\in\mathbb{N}$, $\sigma\in S_n$, and $x_1,\ldots,x_n\in X$,

$$h(\sigma^{\alpha_X}(x_1,\ldots,x_n))=\sigma^{\alpha_Y}(h(x_1),\ldots,h(x_n))$$

If $\alpha,\alpha_X,\alpha_Y$ are clear from context, we omit it from the notation.

We identify $S$ with the polynomial endofunctor $S=\bigsqcup_{n\in\mathbb{N}}S_n\times\mathrm{Id}^n$ on **Set**. Equivalently, an $S$-algebra is as a pair $(X,\alpha)$ consisting of a set $X$ and a function $\alpha\colon SX\to X$. The original definition is obtained by setting

$$\sigma^\alpha(x_1,\ldots,x_n)=\alpha(\sigma,x_1,\ldots,x_n)$$

An $S$-algebra homomorphism is equivalently described as $h\colon(X,\alpha_X)\to(Y,\alpha_Y)$ such

$$\frac{p(0,\ldots,n)=q(0,\ldots,m)\quad x_{(-)}\colon \mathbb{N}\to X}{p(x_0,\ldots,x_n)=q(x_0,\ldots,x_m)} \qquad \text{(Con)}\frac{(\forall i\le n)\ p_i=q_i\quad \sigma\in S_n}{\sigma(p_1,\ldots,p_n)=\sigma(q_1,\ldots,q_n)}$$

**Figure 4.1:** The rules for deriving $\mathsf{EQ}\vdash t=s$ for $p,q,r\in S^*X$, including a generalized version of the inference rule (Con) from Figure 2.1.

that $\alpha_Y\circ h=S(h)\circ\alpha_X$.

Each element $(t(0,\ldots,n),s(0,\ldots,m))\in\mathsf{EQ}$ should be thought of as an equation between $S$-terms, and so we instead write $t\overset{(\mathsf{EQ})}{=}s$. We replace the natural numbers appearing in $S$-terms with $x,x_i,y,y_i$. A formal equation $t=s$ is a *consequence* of EQ, written $\mathsf{EQ}\vdash t=s$, if it can be derived from (Ref), (Sym), and (Tra) from Figure 2.1 and the rules in Figure 4.1 (i.e., equational logic [Bir35]). A binary relation satisfying the rules in Figure 4.1 is an EQ-*congruence*.

**Definition 4.1.5.** An algebraic theory $(S,\mathsf{EQ})$ *presents* a monad $(M,\eta,\mu)$ if there is a natural transformation $\rho\colon SM\Rightarrow M$ such that for any set $X$, $(MX,\rho_X)$ is the free $(S,\mathsf{EQ})$-algebra on $X$. That is, $(MX,\rho_X)$ satisfies EQ, and for any $S$-algebra $(Y,\alpha)$ also satisfying EQ and any function $h\colon X\to Y$, there is a unique $S$-algebra homomorphism $h^\alpha\colon (MX,\rho_X)\to(Y,\alpha)$ such that $h=h^\alpha\circ\eta$.

This universal property implies that any two monads presented by a given algebraic theory are isomorphic, so we often say "the" monad presented by an algebraic theory.

*Remark* 4.1.6. The definition of monad presentation I have chosen to employ makes the algebraic signature portion of an algebraic presentation explicit. A second, much older definition requires that the full subcategory of $\mathrm{Alg}(S)$ consisting of $S$-algebras satisfying EQ is isomorphic to the Eilenberg-Moore category for the monad [Bec69; BSV19; BSV21]. A third definition involves a monad quotient, as seen in [RHE22]. All three are equivalent in the current setting [Man76, Theorem 2.17].

*Example* 4.1.7. The finite powerset functor is part of the monad $(\mathcal{P}_\omega,\{-\},\bigcup)$ that is presented by the theory of semilattices with bottom [Joh82, p. 4.4]. The theory of semilattices is the pair $(\{0\}+\{+\}\times\mathrm{Id}^2,\mathsf{SL})$, since the arity of a constant operation is 0 and $+$ is a binary operation, and SL consists of

$$x+0\overset{(\mathsf{SL1})}{=}x \qquad x+x\overset{(\mathsf{SL2})}{=}x \qquad x+y\overset{(\mathsf{SL3})}{=}y+x \qquad x+(y+z)\overset{(\mathsf{SL4})}{=}(x+y)+z$$

Not every algebraic theory has such a familiar presentation as the theory of semilattices, but it is nevertheless true that every algebraic theory presents a monad. Without a concrete example in mind, i.e., given an abstract algebraic theory $(S, \mathsf{EQ})$, we define the monad $(M, \eta, \mu)$ by setting

$$MX = (S^*X)/\mathsf{EQ} = \{[q]_{\mathsf{EQ}} \mid q \in S^*X\} \qquad [q]_{\mathsf{EQ}} = \{p \in S^*X \mid \mathsf{EQ} \vdash p = q\}$$

That is, $MX$ is the set of EQ-congruence classes of $S$-terms. The unit $\eta$ computes congruence classes of variables, and $\mu$ evaluates terms. This monad is presented by $(S, \mathsf{EQ})$, as witnessed by the transformation $\rho$ defined to be the restriction of $\mu$ to the operations of $S$ on $S$-terms [RHE22]. We take this to be the default presentation of an arbitrary algebraic theory.

*Remark* 4.1.8. In the future, given $p \in S^*X$ we write $p$ instead of $[p]_{\mathsf{EQ}}$ when it is understood from context that an element of $MX$ is being referred to.

Our aim is to develop a (co)algebraic framework for studying $B_M$-coalgebras when $M$ is the functor part of a monad presented by an algebraic theory. Our framework will take the form of a singly-typed syntax (in the style of Milner [Mil84]) for specifying recursive processes, a coalgebraically defined operational interpretation of said processes, and an axiomatization of a general notion of behavioural equivalence between programs.

**Branching theories and unguarded recursion**

As it turns out, capturing arbitrary recursive programs in a singly-typed syntax requires an operational interpretation of unguarded recursive calls. Unguarded recursion (by definition) occurs at the level of branching, and requires some design choices on the part of the user of the framework. In sum, the ingredients needed for one of our process calculi are as follows.

**Definition 4.1.9.** An *(equational) branching theory* is a triple $(S, \mathsf{EQ}, \mathsf{fp})$ consisting of a *nontrivial*[3] equational theory $(S, \mathsf{EQ})$ and a natural transformation $\mathsf{fp} : S^*(1 + \mathrm{Id}) \Rightarrow S^*$, called the *variable reduction operator*, such that whenever $\mathsf{EQ} \vdash t = s$, $\mathsf{EQ} \vdash \mathsf{fp}(t) = \mathsf{fp}(s)$. A branching theory $(S, \mathsf{EQ}, \mathsf{fp})$ is called *iterative* if the variable

[3] That is, the equation $x = y$ is not a consequence of EQ for distinct $x$ and $y$.

reduction operator furthermore satisfies

$$\mathsf{EQ} \vdash \mathsf{fp}_X(t(u,\vec{x})) = t(\mathsf{fp}_X(t(u,\vec{x})),\vec{x}) \tag{4.3}$$

for any $t(u,\vec{x}) \in S^*(1+X)$, where $1 = \{u\}$.

The nontriviality assumption is equivalent to requiring that the unit $\eta$ of the monad presented by $(S,\mathsf{EQ})$ is injective. That is, the EQ-congruence classes $[x]_{\mathsf{EQ}}$ and $[y]_{\mathsf{EQ}}$ in $MX$ are distinct for distinct variables $x$ and $y$ in $X$.

Effectful process calculi allow for the specification of recursive programs like $\mu v\ v$, the program that is recursively defined to be itself. This recursive definition does not have a canonical meaning in general, so to give it meaning requires a design choice. A variable reduction operator represents such a design choice. For example, the divergent program while true do assert true can be modelled as the process term $\mu v\ v$ (see Example 4.2.7). One way to interpret this program is the same way we might interpret assert false (i.e., as a crash). Another valid way to interpret the divergent program is as raise Exception('Infinite loop'), like in Python. This can modelled by including a constant $c$ (representing the exception) in the signature $S$ and using the variable reduction operator $\mathsf{fp}_X(t(u,\vec{x})) = t(c,\vec{x})$.

One should think of the variable reduction operator as an eliminable part of the syntax: Let $X$ be a set of variables containing a distinguished variable $u$. Then where $1 = \{u\}$, $X \cong 1 + X \setminus u$, so fp defines a function $\mathsf{fp} : S^*X \to S^*(X \setminus u)$. Composing with the inclusion $S^*(X \setminus u) \hookrightarrow S^*X$, we obtain an operator on $S^*X$ that produces terms without the free variable $u$.

**Definition 4.1.10.** We write $\mathsf{fp}\,u\ p(u,\vec{x})$ to denote the term $\mathsf{fp}_X(p(u,\vec{x}))$ as it appears in $S^*X$, for any $p(u,\vec{x}) \in S^*X$.

Note that by construction, $\mathsf{fp}\,u\ p(u,\vec{x}) = \mathsf{fp}\,v\ p(v,\vec{x})$ for any $u,v \in X$, and that by naturality $S^*(f)(\mathsf{fp}\,u\ p(u,\vec{x})) = \mathsf{fp}\,u\ p(u,f(\vec{x}))$ for any $f : X \to Y$ such that $u \in X \cap Y$ and $f^{-1}(u) = \{u\}$. That is, the term produced by the variable reduction operator does not depend on the symbol used for the variable or the peripheral variable assignments.

The variable reduction operator fp gives rise to a natural transformation (of the same name) $\mathsf{fp} : M(1+\mathrm{Id}) \Rightarrow M$ on the presented monad. Concretely, if $p \in S^*(1+X)$, the map $\mathsf{fp}_X : M(1+X) \to MX$ acts on congruence classes of terms, i.e.,

$\mathsf{fp}\, u\ [p(u,\vec{x})]_{\mathsf{EQ}} = [\mathsf{fp}\, u\ p(u,\vec{x})]_{\mathsf{EQ}}$ for any $p(u,\vec{x}) \in S^*(1+X)$.

**Examples**

We conclude this section with a number of examples of iterative branching theories and the monads they present.

*Example* 4.1.11. For a fixed finite set $At$ of *atomic tests*, the theory of *guarded algebras* is the pair $(\{0\} + 2^{At} \times \mathrm{Id}^2, \mathsf{GA})$, where GA consists of the equations

$$x +_B x \overset{(\mathsf{GA1})}{=} x \qquad x +_{At} y \overset{(\mathsf{GA2})}{=} x \qquad x +_B y \overset{(\mathsf{GA3})}{=} y +_{\bar{B}} x$$
$$(x +_B y) +_C z \overset{(\mathsf{GA4})}{=} x +_{B\cap C} (y +_C z)$$

Here, $+_B$ is the binary operation associated with the set of atoms $B \subseteq At$, and $\bar{B} = At \setminus B$. The theory of guarded algebras is presented by the monad $((\bot + \mathrm{Id})^{At}, \lambda\alpha.(-), \Delta^*)$, where $(\lambda\alpha.x)(\alpha) = x$ and $\Delta^*(F)(\alpha) = F(\alpha)(\alpha)$. The idea is that $+_B$ acts like an if-then-else clause in an imperative program. This is reflected in a free guarded algebra $((\bot + X)^{At}, \rho_X)$, where for a pair of maps $h_1, h_2 \colon At \to \bot + X$,

$$\rho_X(+_B, h_1, h_2)(\alpha) = \begin{cases} h_1(\alpha) & \text{if } \alpha \in B \\ h_2(\alpha) & \text{otherwise} \end{cases}$$

Equipped with the variable reduction operator $\mathsf{fp}\, u\ t(u,\vec{x}) = t(0,\vec{x})$, the theory of guarded algebra is an iterative branching theory. This can be seen as follows: Given a term $t(u,\vec{x})$[4], we can use (GA3) and (GA4) to find a decomposition of $t(u,\vec{x})$ consisting of a term $s(\vec{x})$ (in which $u$ is not free) and $B \subseteq At$ such that $\mathsf{GA} \vdash t(u,\vec{x}) = u +_B s(\vec{x})$. Inductively, if $t_i(u,\vec{x}) =_{\mathsf{GA}} u +_{B_i} s_i(\vec{x})$ for $i \in \{1,2\}$, then given $C \subseteq At$ and $t(u,\vec{x}) = t_1(u\vec{x}) +_C t_2(u,\vec{x})$,

$$\begin{aligned}
\mathsf{GA} \vdash t_1(u,\vec{x}) +_C t_2(u,\vec{x}) &= (u +_{B_1} s_1(\vec{x})) +_C (u +_{B_2} s_2(\vec{x})) && \text{(ind. hyp.)} \\
&= u +_{B_1\cap C} (s_1(\vec{x}) +_C (u +_{B_2} s_2(\vec{x}))) && \text{(GA4)} \\
&= u +_{B_1\cap C} ((u +_{B_2} s_2(\vec{x})) +_{\bar{C}} s_1(\vec{x})) && \text{(GA3)} \\
&= u +_{B_1\cap C} (u +_{B_2\cap\bar{C}} (s_2(\vec{x}) +_{\bar{C}} s_1(\vec{x}))) && \text{(GA4)}
\end{aligned}$$

[4]We will usually use $p(\vec{x}), q(\vec{x})$ for terms ($p$ for *polynomial*), but in some examples this clashes with standard notation and we use $t(\vec{x}), s(\vec{x})$ instead.

$$= (u +_{B_1 \cap C} u) +_{(B_1 \cap C) \cup (B_2 \cap \bar{C})} (s_2(\vec{x}) +_{\bar{C}} s_1(\vec{x})) \qquad \text{(GA3,GA4)}$$
$$= u +_{(B_1 \cap C) \cup (B_2 \cap \bar{C})} (s_2(\vec{x}) +_{\bar{C}} s_1(\vec{x}))$$

Setting $B = (B_1 \cap C) \cup (B_2 \cap \bar{C})$ and $s(\vec{x}) = s_2(\vec{x}) +_{\bar{C}} s_1(\vec{x})$ gives us such a decomposition. To see that the fixed-point equation (4.3) is satisfied, we compute using the decomposition $t(u,\vec{x}) =_{\mathsf{GA}} u +_B s(\vec{x})$,

$$\begin{aligned}
\mathsf{GA} \vdash t(t(0,\vec{x}),\vec{x}) &= t(0,\vec{x}) +_B s(\vec{x}) && \text{(decomposition, substitution)} \\
&= (0 +_B s(\vec{x})) +_B s(\vec{x}) && \text{(decomposition, congruence)} \\
&= 0 +_B (s(\vec{x}) +_B s(\vec{x})) && \text{(GA4)} \\
&= 0 +_B s(\vec{x}) && \text{(GA1)} \\
&= t(0,\vec{x}) && \text{(decomposition, substitution)}
\end{aligned}$$

Thus, $(\mathsf{GA}, \mathsf{fp})$ is an iterative branching theory.

The theory of guarded algebras dates back to the algebras of if-then-else clauses studied in [McC61; MN87; Man91; BT83; BÉ88]. In particular, guarded algebras are examples of *McCarthy algebras*, introduced by Manes in [Man91].[5]

*Example* 4.1.12. The theory of *pointed convex algebras* [BSV21] is the pair $(\{0\} + [0,1] \times \mathrm{Id}^2, \mathsf{CA})$, where CA consists of the equations

$$x \oplus_p x \overset{(\mathsf{CA1})}{=} x \qquad x \oplus_1 y \overset{(\mathsf{CA2})}{=} x \qquad x \oplus_p y \overset{(\mathsf{CA3})}{=} y \oplus_{\bar{p}} x$$
$$(x \oplus_p y) \oplus_q z \overset{(\mathsf{CA4})}{=} x \oplus_{pq} (y \oplus_{\frac{q\bar{p}}{1-pq}} z)$$

Here, $\oplus_p$ is the binary operation with index $p \in [0,1]$, $\bar{p} = 1 - p$, and $pq \neq 1$. This theory presents the finite subprobability distribution monad $(\mathcal{D}(\bot + \mathrm{Id}), \delta_{(-)}, \sum)$, which is defined to be

$$\mathcal{D}(\bot + X) = \left\{ \theta : X \to [0,1] \;\middle|\; \begin{array}{c} \{x \mid \theta(x) > 0\} \text{ is finite} \\ \sum_{x \in X} \theta(x) \leq 1 \end{array} \right\}$$

[5] More information on the theory of guarded algebras can be found in Appendix A.

for any set $X$, and for any $x \in X$, $\theta \in \mathcal{D}(\bot + X)$, and $\Theta \in \mathcal{D}(\bot + \mathcal{D}(\bot + X))$,

$$\delta_x(y) = \begin{cases} 1 & x = y \\ 0 & \text{otherwise} \end{cases} \qquad \sum(\Theta)(\theta) = \sum_{y \in X} \Theta(\theta) \cdot \theta(y)$$

This is witnessed by the transformation $\rho$ that takes $0$ to the trivial subdistribution and computes the Minkowski sum

$$\rho_X(\oplus_p, \theta, \psi) = p \cdot \theta + (1-p) \cdot \psi$$

for each $p \in [0,1]$, $\theta, \psi \in \mathcal{D}(\bot + X)$. An iterative branching theory can be obtained from the theory of pointed convex semilattices: Its effect on a finitely supported subdistribution $\theta \colon X \to [0,1]$ is given by

$$(\mathsf{fp}\, u\ \theta)(x) = \begin{cases} 0 & x = u \text{ or } \theta(u) = 1 \\ \frac{\theta(x)}{1-\theta(u)} & \text{otherwise} \end{cases}$$

Intuitively, it redistributes the weight of the variable $u$ to the rest of the support. This is indeed iterative: Given terms $t(u,\vec{x}), s(u,\vec{x})$, such that $\theta = [t(u,\vec{x})]_{\mathsf{CA}}$ and $\psi = [s(u,\vec{x})]_{\mathsf{CA}}$, the subdistribution $\theta[\psi/u] := [t(s(u,\vec{x}),\vec{x})]_{\mathsf{CA}}$ is given by

$$\theta[\psi/u](z) = \begin{cases} \theta(u)\ \psi(z) + \theta(z) & u \neq z \\ \theta(u)\ \psi(u) & u = z \end{cases}$$

To check that the fixed-point equation (4.3) is satisfied, there are two cases to consider. If $\theta(u) < 1$, then

$$\begin{aligned} \theta[\mathsf{fp}\, u\ \theta/u](z) &= \begin{cases} \theta(u)\ (\mathsf{fp}\, u\ \theta(z)) + \theta(z) & u \neq z \\ \theta(u)\ (\mathsf{fp}\, u\ \theta(u)) & u = z \end{cases} \\ &= \begin{cases} \theta(u)\ \left(\frac{1}{1-\theta(u)}\theta(z)\right) + \theta(z) & u \neq z \\ \theta(u)\ (0) & u = z \end{cases} \end{aligned}$$

$$= \begin{cases} \frac{\theta(u)}{1-\theta(u)}\theta(z) + \frac{1-\theta(u)}{1-\theta(u)}\theta(z) & u \neq z \\ 0 & u = z \end{cases}$$

$$= \begin{cases} \frac{\theta(u)+1-\theta(u)}{1-\theta(u)}\theta(z) & u \neq z \\ 0 & u = z \end{cases}$$

$$= \begin{cases} \frac{1}{1-\theta(u)}\theta(z) & u \neq z \\ 0 & u = z \end{cases}$$

$$= \mathsf{fp}\, u\ \theta(z)$$

If $\theta(u) = 1$, then $\mathsf{fp}\, u\ \theta = 0$ and $\theta(z) = 0$ for $z \neq u$. Thus,

$$\theta[\mathsf{fp}\, u\ \theta/u](z) = \begin{cases} \theta(u)\ (\mathsf{fp}\, u\ \theta(z)) + \theta(z) & u \neq z \\ \theta(u)\ (\mathsf{fp}\, u\ \theta(u)) & u = z \end{cases} = \begin{cases} \theta(u)\ (0) + 0 & u \neq z \\ \theta(u)\ (0) & u = z \end{cases} = 0$$

*Remark* 4.1.13. The branching theory corresponding to the convex algebra case considered in [Sch22b] had $\mathsf{fp}\, u\ p(x,\vec{y}) = p(0,\vec{y})$. This does *not* define an iterative branching theory. In general, replacing the distinguished variable by a closed term in $S^*X$ produces a variable reduction operator that may not be iterative.

*Example* 4.1.14. The theory of *pointed convex semilattices* studied in [BSV21; VW06; BSS17] combines the theory of semilattices and the theory of convex algebras. It has both a binary operation $+$ mimicking nondeterministic choice and the probabilistic choice operations $\oplus_p$ indexed by $p \in [0,1]$. Formally, it is given by the pair $(1 + \{+\} \times \mathrm{Id}^2 + [0,1] \times \mathrm{Id}^2, \mathsf{CS})$, where $\mathsf{CS}$ is the union of $\mathsf{SL}$, $\mathsf{CA}$, and the distributive law

$$(x+y) \oplus_p z \overset{(\mathsf{D})}{=} (x \oplus_p z) + (y \oplus_p z)$$

This theory presents the pointed convex powerset monad $(\mathcal{C}, \eta^{\mathcal{C}}, \mu^{\mathcal{C}})$, where $\mathcal{C}X$ is the set of finitely generated convex subsets of $\mathcal{D}(\bot + X)$ containing the zero distribution $\delta_\bot$, and for $x \in X$ and $Q \in \mathcal{C}\mathcal{C}X$,

$$\eta^{\mathcal{C}}(x) = \{p \cdot \delta_x \mid p \in [0,1]\} \qquad \mu^{\mathcal{C}}(Q) = \bigcup_{\Theta \in Q} \left\{ \sum_{U \in \mathcal{C}_0 X} \Theta(U) \cdot \theta_U \;\middle|\; (\forall U \in \mathcal{C}X)\ \theta_U \in U \right\}$$

The witnessing transformation $\rho^{\mathcal{C}}$ takes 0 to $\{\delta_\perp\}$, computes the set-wise Minkowski sum in place of $\oplus_p$, and interprets the $+$ operation as the convex hull of the union $\rho^{\mathcal{C}}_X(+,U,V) = \mathrm{Conv}(U \cup V)$, where

$$\mathrm{Conv}(U) = \left\{ \sum_{i=1}^{n} p_i \cdot \theta_i \;\middle|\; p_i \in [0,1], \theta_i \in U, \text{ and } \sum p_i \leq 1 \right\}$$

There is an intuitive variable reduction operator $\mathsf{fp} : \mathcal{C}(1+\mathrm{Id}) \Rightarrow \mathcal{C}$ that turns the theory of pointed convex semilattices into an iterative branching theory, essentially given by lifting the iterative variable reduction operator from the theory of convex algebra. More concretely, given a convex set $U = \mathrm{Conv}(r_0\delta_u, r_1\delta_u + \theta_1, \ldots, r_n\delta_u + \theta_n)$ where $r_1, \ldots, r_n < 1$ and $\theta_1(u) = \cdots = \theta_n(u) = 0$,

$$\mathsf{fp}\, u\, U = \mathrm{Conv}\left( \frac{1}{1-r_1}\theta_1, \ldots, \frac{1}{1-r_n}\theta_n \right)$$

This clearly defines a variable reduction operator. To see that it is iterative, begin with the observation that substitution operates as follows: Given $U = [t(u,\vec{x})]_{\mathsf{CS}}$ and $V = [s(u,\vec{x})]_{\mathsf{CS}}$, the convex set $U[V/u] := [t(s(u,\vec{x}),\vec{x})]_{\mathsf{CS}}$ is given by

$$U[V/u] = \mathrm{Conv}(\{\theta[\psi/u] \mid \theta \in U \text{ and } \psi \in V\})$$

Computing $U[\mathsf{fp}\, u\, U/u]$, we obtain

$$\begin{aligned} U[\mathsf{fp}\, u\, U/u] &= \mathrm{Conv}(\{\theta[\psi/u] \mid \theta \in U \text{ and } \psi \in \mathsf{fp}\, u\, U\}) \\ &= \mathrm{Conv}(r_0(\mathsf{fp}\, u\, U) \cup \{r_i\psi + \theta_i \mid 0 < i \leq n \text{ and } \psi \in \mathsf{fp}\, u\, U\}) \\ &= \mathrm{Conv}(\{r_i\psi + \theta_i \mid 0 < i \leq n \text{ and } \psi \in \mathsf{fp}\, u\, U\}) \end{aligned}$$

The third equality follows from the identity

$$r_i\left(\frac{1}{1-r_i}\theta_i\right) + \theta_i = \frac{1}{1-r_i}\theta_i$$

because the identity above tells us that every $r_0(1-r_i)^{-1}\theta_i$ in $r_0(\mathsf{fp}\, u\, U)$ can be written $r_0(r_i(1-r_i)^{-1}\theta_i + \theta_i)$. For a similar reason, this identity also implies that $\mathsf{fp}\, u\, U \subseteq U[\mathsf{fp}\, u\, U/u]$. Now, to prove that $\mathsf{fp}\, u$ is an iterative variable reduction operator, we need to establish that $U[\mathsf{fp}\, u\, U] \subseteq \mathsf{fp}\, u\, U$. Given $\psi = \sum_{j=1}^{n} p_j \frac{1}{1-r_j}\theta_j$ for

$\sum p_j \leq 1$, observe that

$$r_i\psi + \theta_i = \sum_{j=1}^{n}\left(p_j\frac{r_i}{1-r_j}\theta_j\right) + \theta_i = r_i\sum_{i\neq j}\left(p_j\frac{1}{1-r_j}\theta_j\right) + (1-r_i(1-p_i))\frac{1}{1-r_i}\theta_i$$

A bit of calculation reveals that since $\sum p_j \leq 1$, $r_i\sum_{i\neq j} p_j + 1 - r_i(1-p_i) \leq 1$. Therefore,

$$r_i\psi + \theta_i \in \mathrm{Conv}\left(r_1\frac{1}{1-r_1}\theta_1 + \theta_1, \ldots, r_n\frac{1}{1-r_n}\theta_n + \theta_n\right) = \mathsf{fp}\, u\, U$$

and it follows that $U[\mathsf{fp}\, u\, U] \subseteq \mathsf{fp}\, u\, U$. Thus, $\mathsf{fp}\, u\, U = U[\mathsf{fp}\, u\, U/u]$, as desired.

*Example* 4.1.15. The theory of *convex semilattices with top* [MOW03; BSV19] is the theory of pointed convex semilattices with two alterations: The Minkowski sum $\oplus_p$ must have $p \in (0,1)$, and there is an additional axiom $x \oplus_r 0 \overset{(\mathsf{T})}{=} 0$. The theory of convex semilattices with top is written $\mathsf{CS}\top$. The monad presented by this theory is $(\mathcal{C}_\top, \eta^{\mathcal{C}_\top}, \mu^{\mathcal{C}_\top})$, where $\mathcal{C}_\top X$ is the set of finitely generated convex subsets of $\mathcal{D}X$ (including $\emptyset$). Unlike the previous example, $0$ represents $\emptyset$ in the free convex semilattice with top (hence, $x \oplus_r 0 = 0$). A similar variable reduction operator to the one in the convex semilattices example turns $\mathsf{CS}\top$ into an iterative branching theory.

*Remark* 4.1.16. There are many examples of equational theories that do not admit any iterative branching structure. One non-trivial example is the equational theory of semigroups, which consists of a single binary operation $\cdot$ and the equation $x\cdot(y\cdot z) = (x\cdot y)\cdot z$. The free semigroup is the non-empty list functor $X \mapsto X^+$. This does not admit an iterative branching structure for the following reason: Suppose that $\mathsf{fp}$ were an iterative variable reduction operator for this theory, and let $X$ be a set with $u, x \in X$. Then we could prove the equation $\mathsf{fp}\, u\ (u\cdot x) = (\mathsf{fp}\, u\ (u\cdot x))\cdot x$, by (4.3). No word in $X^+$ can satisfy the equation $w = w\cdot x$ (words have a finite length), so in particular $\mathsf{fp}\, u\ (u\cdot x)$ cannot satisfy this equation either. Thus, there is no iterative branching structure on the equational theory of semigroups.

## 4.2 Specifications of Processes

Fix a branching theory $(S, \mathsf{EQ}, \mathsf{fp})$ presenting a monad $(M, \eta, \mu)$ on **Set**, as well as two sets *Var* and *Act* of output variables and action symbols. Recall the endofunctor defined $B_M = M(Var + Act \times \mathrm{Id})$ on **Set**.

**Definition 4.2.1.** An *effectful transition system* is a $B_M$-coalgebra. An *effectful process*

is a state of an effectful transition system. An effectful process is *finite* if it is a state in a finite effectful transition (sub)system.

In this section, we give a syntactic and uniformly defined specification system for effectful processes along with an operational semantics inspired by Brzozowski's derivatives [Brz64]. We are primarily concerned with the specifications of finite processes, and indeed the process terms we construct below denote processes with finitely many states. The converse is also true, that every finite $B_M$-coalgebra admits a specification in the form of a process term, but we defer this result to Section 4.4 because of its relevance to the completeness theorem there.

**Definition 4.2.2.** Given a set *Var* of *output variables*, a set *Act* of *action symbols*, and an algebraic signature $S$, the set *Exp* of *process terms*, is given by

$$e, e_i ::= v \mid \sigma(e_1, \ldots, e_n) \mid ae \mid \mu v\ e$$

where $v \in Var$, $a \in Act$, and $\sigma \in S_n$.

The process $\sigma(e_1, \ldots, e_n)$ is the process that branches into $e_1, \ldots, e_n$ using an $n$-ary operation $\sigma$ as the branching constructor. The expression $ae$ denotes the process that performs the action $a$ and then proceeds with $e$. Following [Mil84], we use output variables in one of two ways, depending on the context in which they appear: A variable $v$ appearing in $e$ is *free* if it does not appear within the scope of $\mu v$ and *bound* otherwise. If $v$ is free in $e$, then $v$ denotes output $v$. Otherwise, $v$ denotes a *goto* statement that returns the process to the $\mu v$ that binds $v$. Finally, as a convention, we use the symbol $0$ to denote any expression of the form $\mathsf{fp}\, u\ u$, which essentially denotes the *deadlock* process that takes no action.

Abstractly, process terms form the initial $\Sigma_M$-algebra $(Exp, \mathsf{ev})$, where $\Sigma_M$ is the polynomial endofunctor defined by

$$\Sigma_M = S + Var + Act \times \mathrm{Id} + \{\mu\} \times Var \times \mathrm{Id}$$

and the algebra map $\mathsf{ev} \colon \Sigma_M Exp \to Exp$ evaluates $\Sigma_M$-terms.

**Small-step Semantics**

Next we give a small-step semantics to process terms that is uniformly defined for the process types in our parametrized family. Most algebraic theories lack a familiar presentation, which ultimately prevents the corresponding semantics from taking the traditional form of a set of inference rules describing transition relations. Instead, we take an abstract approach by directly defining a $B_M$-coalgebra structure $\varepsilon : Exp \to B_M Exp$ that mirrors the intuitive descriptions of the executions of process terms above. The formal description of $\varepsilon$ is summarized in Figure 4.2, although it requires further explanation.

Before we see the formal definition of $\varepsilon(\mu v\ e)$, in particular, let us stop to think about how it might work. Intuitively, $\mu v\ e$ carries out the process denoted by $e$ until it reaches an $\mathsf{output}\ v$, at which point it ignores the output and loops back to the beginning. If we imagine that a process term executes "from left-to-right", the behaviour just described can be captured by setting $\varepsilon(\mu v\ e) = \varepsilon(e)[\mu v\ e//v]$, where the substitution operator $[\mu v\ e//v]$ replaces each pair $(a,g)$ appearing in $\varepsilon(e)$ with $(a, g[e/v])$, and $[e/v]$ is the syntactic substitution operator that replaces free instances of the variable $v$ in $g$ with $\mu v\ e$ (these operators are defined formally in Definition 4.2.4). In other words, $\mu v\ e$ repeatedly performs all the actions $e$ performs.

However, this only accurately describes recursion in $v$ when in every branch of its execution, $e$ performs an action before outputting $v$. For example, the process $\mu v\ v$ never outputs $v$ and never performs any action at all, even though $\varepsilon(v)[\mu v\ v//v] = \varepsilon(v) =$ "$\mathsf{output}\ v$". We deal with this as follows: If an $\mathsf{output}\ v$ is immediately reached by a branch of $e$, then we *iterate out* the variable $v$ in the term representation of $\varepsilon(e)$ before we apply $[\mu v\ e//v]$. Formally, we set $\varepsilon(\mu v\ e) = \mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]$, where $\mathsf{fp}\ v$ is the iterative variable reduction operator. For example, the term $\mu v\ v$ represents an $S$-term with no free variables, so $\varepsilon(\mu v\ v) = \mathsf{fp}\ v\ \varepsilon(v)[\mu v\ v//v] = \mathsf{fp}\ v\ v = 0$.

In Definition 4.2.3 and Definition 4.2.4, we formally define the substitution operators necessary for giving a precise description of $\varepsilon$.

**Definition 4.2.3.** Let $e \in Exp$. Write $\mathrm{fv}(e)$ for the set of variables that appear free in a process term $e$. The set of variables *guarded in* $e$ is written $\mathrm{gv}(e)$ and defined

$$\mathrm{gv}(v) = Var \setminus v \quad \mathrm{gv}(\sigma(e_1, \ldots, e_n)) = \bigcap_{i \leq n} \mathrm{gv}(e_i)$$

$$\mathrm{gv}(ae) = \mathrm{fv}(e) \quad \mathrm{gv}(\mu v\ e) = \{v\} \cup \mathrm{gv}(e)$$

If $v \notin \mathrm{gv}(e)$, then $v$ appears *unguarded* in $e$.

Note that if $v$ is not free in $e$, then $v \in \mathrm{gv}(e)$.

We allow for recursion in unguarded variables by baking the variable reduction operator into the operational semantics. Formally, this requires us to define *syntactic* and *guarded syntactic substitution operators*.

**Definition 4.2.4.** Given a process term $g$ and a variable $v \in \mathit{Var}$, the *syntactic substitution of $v$ for $g$* is the partial function $[g/v] \colon \mathit{Exp} \rightharpoonup \mathit{Exp}$ defined

$$u[g/v] = \begin{cases} g & u = v \\ u & \text{otherwise} \end{cases} \qquad \sigma(e_1, \ldots, e_n)[g/v] = \sigma(e_1[g/v], \ldots, e_n[g/v])$$

$$(ae)[g/v] = ae[g/v] \qquad (\mu u\ e)[g/v] = \begin{cases} \mu u\ e & u = v \\ \mu u\ e[g/v] & u \neq v \text{ and } u \notin \mathrm{fv}(g) \\ \text{undefined} & \text{otherwise} \end{cases}$$

Analogously, the *guarded syntactic substitution of $v$ for $g$* is the partial $S$-algebra homomorphism $[g//v] \colon B_M\mathit{Exp} \rightharpoonup B_M\mathit{Exp}$ defined recursively by

$$u[g//v] = u \qquad (a, f)[g//v] = (a, f[g/v])$$
$$\sigma(p_1, \ldots, p_n)[g//v] = \sigma(p_1[g//v], \ldots, p_n[g//v])$$

Syntactic substitution replaces free instances of $v$ with $g$ without binding variables in $g$, and guarded syntactic substitution substitutes instances of guarded variables but leaves unguarded variables alone.

Both syntactic substitution operators are only partially defined. The following lemma characterizes expressions in the domain of each substitution operator.

**Lemma 4.2.5.** *Let $e, g \in \mathit{Exp}$ and $v \in \mathit{Var}$. Then $e[g/v]$ and $\varepsilon(e)[g//v]$ are defined if and only if no variable bound in $e$ appears free in $g$.*

We now have all of the formal notions we need to give a precise definition of the small-step semantics of *Exp*.

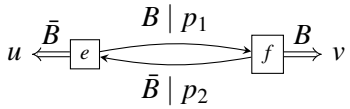
$$\varepsilon(v) = v \qquad \varepsilon(\sigma(e_1,\dots,e_n)) = \sigma(\varepsilon(e_1),\dots,\varepsilon(e_n))$$
$$\varepsilon(ae) = (a,e) \qquad \varepsilon(\mu v\ e) = \mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]$$

**Figure 4.2:** Operational semantics of process terms. Here, $v \in Var$, $a \in Act$, and $e, e_i \in Exp$.

**Definition 4.2.6.** Given an iterative branching theory $(S, \mathsf{EQ}, \mathsf{fp})$ presenting a monad $(M, \eta, \mu)$, the *small-step semantics* of its corresponding set of process terms $Exp$ is the $B_M$-coalgebra $(Exp, \varepsilon)$ defined in Figure 4.2.

Formally, $\varepsilon$ assigns to each process term $e$ an EQ-congruence class $\varepsilon(e)$ of terms from $S^*(Var + Act \times Exp)$. A term from $S^*(Var + Act \times Exp)$ is a combination of variables $v$ and transition-like pairs $(a, e_i)$, so there is often only a small conceptual leap from the coalgebra structure $\varepsilon$ to a more traditional representation of transitions as decorated arrows. We provide the following examples as illustrations of this phenomenon, as well as the specification languages and operational semantics of terms defined above.

*Example* 4.2.7. The *algebra of control flows*, or ACF, is obtained from the iterative branching theory of guarded algebras of Example 4.1.11 and $M = (\bot + \mathrm{Id})^{At}$. To match with the notation in Chapter 3, we will temporarily use $p, q$ for actions (i.e., $Act = \Sigma$ from Chapter 3) and $G = B_M$. Given a structure map $\beta : X \to GX$ and $B \subseteq At$, write $x \xrightarrow{B|p} y$ if $\beta(x)(\alpha) = (p, y)$ for all $\alpha \in B$, and $x \overset{B}{\Rightarrow} v$ if $\beta(x)(\alpha) = v$ for all $\alpha \in B$. The operational semantics returns the constant map $\lambda\alpha.v$ given a variable $v \in Var$ and interprets conditional choice as guarded union.

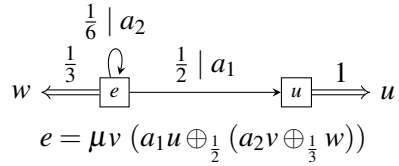



$$e = \mu w\ (p_1(v +_B p_2 w) +_B u)$$
$$f = v +_B p_2 e$$

$$\begin{aligned}
\varepsilon(e) &= \varepsilon(p_1(v +_B p_2 w) +_B u)[e//w] \\
&= (\lambda\alpha.(p_1, v +_B p_2 w) +_B \lambda\alpha.u)[e//w] \\
&= (\lambda\alpha.(p_1, v +_B p_2 w))[e//w] +_B \lambda\alpha.u \\
&= \lambda\alpha.(p_1, v +_B p_2 e) +_B \lambda\alpha.u \\
&= \lambda\alpha.(p_1, f) +_B \lambda\alpha.u
\end{aligned}$$

*Example* 4.2.8. The *algebra of probabilistic actions*, or APA, is obtained from the iterative branching theory of pointed convex algebras in Example 4.1.12 and with $M = \mathcal{D}(\bot + \mathrm{Id})$. For a structure map $\beta : X \to B_{\mathcal{D}(1+\mathrm{Id})}X$, write $x \xrightarrow{k|a} y$ when $\beta(x)(a, y) = k$ and $e \overset{k}{\Rightarrow} v$ when $\beta(e)(v) = k$. The operational semantics returns the Dirac distribution

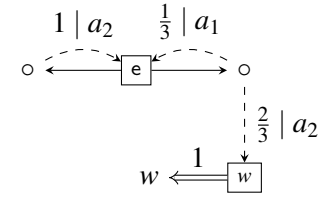
$\delta_v$ for $v \in Var$ and interprets probabilistic choice as the Minkowski sum.




$$e = \mu v\ (a_1 u \oplus_{\frac{1}{2}} (a_2 v \oplus_{\frac{1}{3}} w))$$

$$\begin{aligned}\varepsilon(e) &= \varepsilon(a_1 u \oplus_{\frac{1}{2}} (a_2 v \oplus_{\frac{1}{3}} w))[e//v] \\ &= \left(\frac{1}{2}\delta_{(a_1,u)} + \frac{1}{6}\delta_{(a_2,v)} + \frac{1}{3}\delta_w\right)[e//v] \\ &= \frac{1}{2}\delta_{(a_1,u)} + \frac{1}{6}\delta_{(a_2,e)} + \frac{1}{3}\delta_w\end{aligned}$$

*Example* 4.2.9. The *algebra of nondeterministic probabilistic actions*, or ARB, is obtained from the theory of pointed convex semilattices of Example 4.1.14. For a structure map $\beta : X \to B_{\mathcal{C}}X$, write $x \longrightarrow \circ \overset{k \mid a}{\dashrightarrow} y$ to mean there is a $\theta \in \beta(x)$ such that $\theta(a,y) = k$, and $x \overset{k}{\Rightarrow} v$ to mean there is a $\theta \in \beta(x)$ with $\theta(v) = k$. The operational semantics returns $\eta^{\mathcal{C}}(v)$ given $v \in Var$, interprets nondeterministic choice as convex union, and replaces probabilistic choice with Minkowski sum.




$$e = \mu v\ ((a_1 v \oplus_{\frac{1}{3}} a_2 w) + a_2 v)$$

$$\begin{aligned}\varepsilon(e) &= \varepsilon((a_1 v \oplus_{\frac{1}{3}} a_2 w) + a_2 v)[e//v] \\ &= \mathrm{Conv}\{\varepsilon(a_1 v \oplus_{\frac{1}{3}} a_2 w), \delta_{(a_2,v)}\}[e//v] \\ &= \mathrm{Conv}\left\{\frac{1}{3}\delta_{(a_1,v)} + \frac{2}{3}\delta_{(a_2,w)}, \delta_{(a_2,v)}\right\}[e//v] \\ &= \mathrm{Conv}\left\{\frac{1}{3}\delta_{(a_1,e)} + \frac{2}{3}\delta_{(a_2,w)}, \delta_{(a_2,e)}\right\}\end{aligned}$$

## 4.3 Behavioural Equivalence and the Final Coalgebra

It follows from general considerations that the functor $B_M$ admits a final coalgebra (see Definition 2.5.1): Since $M$ is constructed from a finitary algebraic signature, the coalgebraic signature $B_M$ is *finitary* (a.k.a. *bounded*), meaning it preserves directed colimits [AR94]. Every finitary functor $B$ admits a final coalgebra [Rut00; Adá05], so $B_M$ admits a final coalgebra. In particular, the universal property of the final $B_M$-coalgebra produces the coalgebra homomorphism $!_\varepsilon : (Exp, \varepsilon) \to (Z, \zeta)$, called the *operational semantics*.

In this section, we relate the operational semantics, arising from the coalgebra structure on $Exp$, to a denotational semantics induced by a suitable algebra structure on the domain of *process behaviours*, elements of the final $B_M$-coalgebra.

**Definition 4.3.1** (Operational Semantics [TR98])**.** Given $e \in Exp$, the behaviour $!_\varepsilon(e)$ is called the *final (coalgebra) semantics*, or *operational semantics* of $e$.

For example, the final $B_{\mathcal{P}_\omega}$-coalgebra consists of bisimulation equivalence classes of finite and infinite labelled trees of a certain form [Bar93]. In this setting, $(Exp, \varepsilon)$ is a labelled transition system and the operational semantics $!_\varepsilon$ constructs a tree from a process term by unrolling. Intuitively, this captures the behaviour of a specification by encoding all possible actions and outputs at each time-step in its execution.

**Denotational Semantics**

In addition to forming the state space of the final $B_M$-coalgebra, the set of process behaviours also carries the structure of a $\Sigma_M$-algebra $(Z, \gamma)$. The construction of this $\Sigma_M$-algebra structure requires coalgebraic analogues of syntactic and guarded syntactic substitution from Section 4.2.

**Definition 4.3.2.** For a given behaviour $s \in Z$ and a variable $v \in Var$, the *behavioural substitution* of $s$ for $v$ is the map $\{s/v\} \colon Z \to Z$ defined *corecursively* by the identity

$$\zeta(t\{s/v\}) = \begin{cases} \zeta(s) & \zeta(t) = v \\ u & \zeta(t) = u \neq v \\ (a, r\{s/v\}) & \zeta(t) = (a, r) \\ \sigma(\zeta(t_1\{s/v\}), \ldots, \zeta(t_n\{s/v\})) & \zeta(t) = \sigma(\zeta(t_1), \ldots, \zeta(t_n)) \end{cases} \tag{4.4}$$

for any $t \in Z$. In other words, we define the coalgebra structure $k \colon Z + Z \times \{\bullet\} \to B_M(Z + Z \times \{\bullet\})$, where $k(t) = \zeta(t)$ and

$$k(t\bullet) = \begin{cases} \zeta(s) & \zeta(t) = v \\ u & \zeta(t) = u \neq v \\ (a, r\bullet) & \zeta(t) = (a, r) \\ \sigma(k(t_1\bullet), \ldots, k(t_n\bullet)) & \zeta(t) = \sigma(\zeta(t_1), \ldots, \zeta(t_n)) \end{cases}$$

The identity (4.4) above amounts to the statement that $\{s/v\}$ is the composition of the injection $Z \times \{\bullet\} \hookrightarrow Z + Z \times \{\bullet\}$ with the unique coalgebra homomorphism $!_k \colon (Z + Z \times \{\bullet\}, k) \to (Z, \zeta)$.

The second form of substitution we will make use of is *guarded behavioural substitution*, constructed in analogy with guarded syntactic substitution.

**Definition 4.3.3.** Given $s, v$, the *guarded behavioural substitution operator* $\{s//v\} : B_M Z \to B_M Z$ is defined recursively by

$$u\{s//v\} = u \qquad (a,r)\{s//v\} = (a, r\{s/v\})$$
$$\sigma(r_1,\dots,r_n)\{s//v\} = \sigma(r_1\{s//v\},\dots,r_n\{s//v\})$$

where $u \in Var$, $a \in Act$, and $r, r_i \in Z$ for $i \leq n$.

We define the $\Sigma_M$-algebra structure $(Z,\gamma)$ using a certain system of *behavioural differential equations* [Rut98]. This is the content of Theorem 4.3.4 below.

**Theorem 4.3.4.** *There is a unique $\Sigma_M$-algebra structure $\gamma : \Sigma_M Z \to Z$ such that*

$$\begin{aligned} \zeta(\gamma(v)) &= v & \zeta(\gamma(\sigma,t_1,\dots,t_n)) &= \sigma(\zeta(t_1),\dots,\zeta(t_n)) \\ \zeta(\gamma(a,t)) &= (a,t) & \zeta(\gamma(\mu v,t)) &= \mathsf{fp}\, v\ \zeta(t)\{\gamma(\mu v,t)//v\} \end{aligned} \tag{4.5}$$

*Above, $v \in Var$, $a \in Act$, $t, t_i \in Z$ for $i \leq n$, and $\sigma$ is an $n$-ary operation from $S$.*

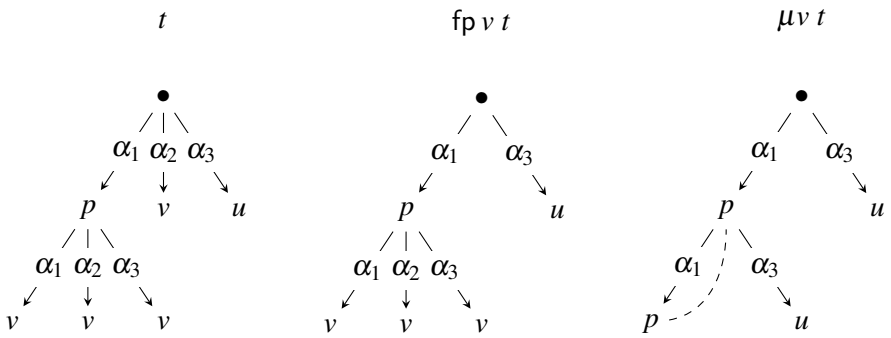
*Proof sketch.* Recall that $\Sigma_M Z = SZ + Var + Act \times Z + \{\mu\} \times Var \times Z$, where the components listed correspond to the algebraic operations in $S$, the output variables, prefixing by an action, and recursion in a variable respectively. By Lambek's lemma [Lam68], $\zeta : Z \to B_M Z$ is a bijection, so the first three equations in (4.5) determine a map $\gamma_{base} : SZ + Var + Act \times Z \to Z$. To obtain $\gamma = [\gamma_{base}, \gamma_{rec}]$, we are left with defining $\gamma_{rec} : \{\mu\} \times Var \times Z \to Z$. The formal construction of the recursion operations $\gamma_{rec}$ is the subject of Section 4.3.1. Roughly, the behaviour $\gamma(\mu v,t)$ is the EQ-equivalence class of a certain coterm (i.e., infinite syntax tree) for the functor $S^*(Var + Act \times \mathrm{Id})$. □

Given $a \in Act$, $t, t_1, \dots, t_n \in Z$, $\sigma \in S_n$, we typically write $at$ for $\gamma(a,t)$, $\sigma(t_1,\dots,t_n)$ for $\gamma(\sigma,t_1,\dots,t_n)$, and $\mu v\, t$ for $\gamma(\mu v,t)$.

*Example* 4.3.5. To see how the recursion operator $\gamma(\mu v,-)$ acts on behaviours, recall from Section 3.1 that the final $(2 + \Sigma \times \mathrm{Id})^{At}$-coalgebra is isomorphic to a set of behaviour trees, partial functions of the form $t : At^+ \rightharpoonup 2 + \Sigma$ (where $\Sigma = Act$ here). For similar reasons, the final $B_M$-coalgebra for $M = (\bot + \mathrm{Id})^{At}$—like in Example 4.2.7—is isomorphic to a set of partial maps $t : At^+ \rightharpoonup \bot + Var + \Sigma$, also visualized as trees[6]. Suppose $At = \{\alpha_1, \alpha_2, \alpha_3\}$. Below is a visualization of the behaviour $t$ of the expression

[6] Compare also with the construction of a final coalgebra consiting of *coterms* in Section 4.3.1

$e = pv +_{\alpha_1} (v +_{\alpha_2} u)$, a visualization of the behaviour $\mathsf{fp}\, v\ t$ defined by $\zeta(\mathsf{fp}\, v\ t) = \mathsf{fp}\, v\ \zeta(t)$, and a visualization of the behaviour $\gamma(\mu v, t)$.




Above, the dashed line indicates that the two subtrees are isomorphic. Notice how the iterative variable reduction operator removes the output $v$ command from only the initial branching of $t$, whereas there are no instances of output $v$ in $\mu v\, t$. Essentially, the recursion operator $\mu v$ corecursively pastes the initial $\alpha$ of the tree $\mathsf{fp}\, v\ t$ wherever it sees a subtree of $t$ with a leaf labelled $v$.

The structure $(Exp, \mathsf{ev})$ is the *initial* $\Sigma_M$-algebra, so there is a unique algebra homomorphism $[\![-]\!] : (Exp, \mathsf{ev}) \to (Z, \gamma)$.

**Definition 4.3.6** (Denotational Semantics [Gog+77])**.** The behaviour $[\![e]\!]$ is called the *initial (algebra) semantics*, or *denotational semantics* of $e$.

The equations in (4.5) characterizing the algebra structure $\gamma\colon \Sigma_M Z \to Z$ of $(Z, \gamma)$ can be seen as a rehashing of the programming constructs of the language $Exp$ that mimics the operational semantics of process terms. The basic constructs are the content of the first three equations in (4.5): Output variables are evaluated so as to behave like the variables of $(Exp, \varepsilon)$, the behaviour $at$ performs $a$ and moves on to $t$, and $\sigma(t_1, \ldots, t_n)$ branches into the behaviours $t_1, \ldots, t_n$ with additional structure determined by the operation $\sigma$.

Behavioural and guarded behavioural substitution interact well with syntactic substitution, as the following two lemmas show. The proof of the first of the two lemmas is long, so for the sake of presentation it has been given its own Section 4.3.2.

**Lemma 4.3.7.** *For any $e, g \in Exp$ such that no free variable of $g$ appears bound in $e$, and for any $v \in Var$, $[\![e[g/v]]\!] = [\![e]\!]\{[\![g]\!]/v\}$.*

*Proof.* See Section 4.3.2. □

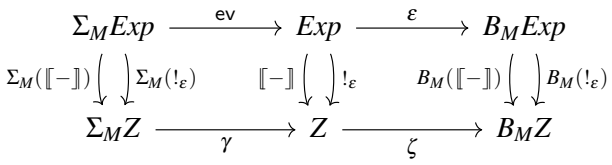



**Figure 4.3:** A diagram of Theorem 4.3.9.

The most important property of the behaviour $\gamma(\mu v,t)$ is that it is the unique solution to $\zeta(r) = \mathsf{fp}\, v\ \zeta(t)\{r//v\}$ in the indeterminate $r$.

**Lemma 4.3.8.** *Let $\llbracket - \rrbracket$ be the unique algebra homomorphism $(Exp, \mathsf{ev}) \to (Z, \gamma)$. For any $g \in Exp$, $p \in B_M Exp$, and $v \in Var$, $B_M(\llbracket - \rrbracket)(p[g//v]) = B_M(\llbracket - \rrbracket)(p)\{\llbracket g \rrbracket //v\}$.*

*Proof.* By induction on $p$. In the base case, we have

$$B_M(\llbracket - \rrbracket)(u[g//v]) = B_M(\llbracket - \rrbracket)(u) = u = u\{\llbracket g \rrbracket //v\} = B_M(\llbracket - \rrbracket)(u)\{\llbracket g \rrbracket //v\}$$

for $u \in Var$. Given $e \in Exp$ and $a \in Act$,

$$\begin{aligned} B_M(\llbracket - \rrbracket)((a,e)[g//v]) &= B_M(\llbracket - \rrbracket)((a, e[g/v])) \\ &= (a, \llbracket e[g/v] \rrbracket) \\ &= (a, \llbracket e \rrbracket \{g/v\}) && \text{(Lemma 4.3.7)} \\ &= (a, \llbracket e \rrbracket)\{g//v\} \\ &= B_M(\llbracket - \rrbracket)((a,e))\{\llbracket g \rrbracket //v\} \end{aligned}$$

The inductive step is straightforward. □

Both the operational semantics and the denotational semantics give intuitively correct meanings to process terms. As one might hope, the two coincide. In other words, the final coalgebra semantics given with respect to operational rules in $Exp$ is equal to the initial algebra semantics given with respect to the programming constructs in $Z$.

**Theorem 4.3.9.** *For any process term $e \in Exp$, we have $!_\varepsilon(e) = \llbracket e \rrbracket$.*

*Proof.* We would like to check that the whole of the diagram in Figure 4.3 commutes. By the universal property of the final coalgebra, it suffices to check that $\llbracket - \rrbracket$ is

a coalgebra homomorphism—that for any $e \in Exp$, $\zeta(\llbracket e \rrbracket) = B_M(\llbracket - \rrbracket) \circ \varepsilon(e)$. This equation can be derived inductively on $e$. Before we proceed, observe that the action of $B_M$ on $\llbracket - \rrbracket$ is the extension of the substitution $(a,e) \mapsto (a, \llbracket a \rrbracket)$ to the unique $S$-algebra homomorphism $B_M Exp \to B_M Z$ that maps output variables to themselves.

For the base case, let $v \in Var$ and observe that by (4.5) and the fact that $\llbracket - \rrbracket$ is a $\Sigma$-algebra homomorphism,

$$\zeta(\llbracket v \rrbracket) = \zeta \circ \gamma(v) = v = B_M(\llbracket - \rrbracket) \circ \varepsilon(v)$$

For the inductive step, assume that for $g \in \{e, e_i\}$, $\zeta(\llbracket g \rrbracket) = B_M(\llbracket - \rrbracket) \circ \varepsilon(g)$. Given $a \in Act$, we have

$$\begin{aligned}
\zeta(\llbracket ae \rrbracket) &= \zeta \circ \gamma(a, \llbracket e \rrbracket) && (\llbracket - \rrbracket \text{ alg. homom.}) \\
&= (a, \llbracket e \rrbracket) && ((4.5)) \\
&= B_M(\llbracket - \rrbracket)(a,e) && (\text{def } B_M.) \\
&= B_M(\llbracket - \rrbracket) \circ \varepsilon(ae) && (\text{Figure 4.2})
\end{aligned}$$

Given $\sigma \in S_n$, we have

$$\begin{aligned}
\zeta(\llbracket \sigma(e_1, \ldots, e_n) \rrbracket) &= \zeta \circ \llbracket - \rrbracket \circ \mathsf{ev}(\sigma, e_1, \ldots, e_n) \\
&= \zeta \circ \gamma \circ \Sigma_M(\llbracket - \rrbracket)(\sigma, e_1, \ldots, e_n) && (\llbracket - \rrbracket \text{ alg. homom.}) \\
&= \zeta \circ \gamma(\sigma, \llbracket e_1 \rrbracket, \ldots, \llbracket e_n \rrbracket) \\
&= \sigma(\zeta(\llbracket e_1 \rrbracket), \ldots, \zeta(\llbracket e_n \rrbracket)) && ((4.5)) \\
&= \sigma(B_M(\llbracket - \rrbracket) \circ \varepsilon(e_1), \ldots, B_M(\llbracket - \rrbracket) \circ \varepsilon(e_n)) && (\text{induct. hyp.}) \\
&= B_M(\llbracket - \rrbracket)(\sigma(\varepsilon(e_1), \ldots, \varepsilon(e_n))) \\
&= B_M(\llbracket - \rrbracket) \circ \varepsilon(\sigma(e_1, \ldots, e_n)) && (\text{Figure 4.2})
\end{aligned}$$

Finally, for $v \in Var$,

$$\begin{aligned}
\zeta(\llbracket \mu v\ e \rrbracket) &= \zeta \circ \llbracket - \rrbracket \circ \mathsf{ev}(\mu v, e) \\
&= \zeta \circ \gamma \circ \Sigma_M(\llbracket - \rrbracket)(\mu v, e) && (\llbracket - \rrbracket \text{ alg. homom.}) \\
&= \zeta \circ \gamma(\mu v, \llbracket e \rrbracket)
\end{aligned}$$

$$
\begin{aligned}
&= \mathsf{fp}\, v\ \zeta([\![e]\!])\{\gamma(\mu v, [\![e]\!])//v\} && \text{(Figure 4.2)}\\
&= \mathsf{fp}\, v\ \zeta([\![e]\!])\{[\![\mu v\ e]\!]//v\} &&\\
&= \mathsf{fp}\, v\ (B([\![-]\!])\circ\varepsilon(e))\{[\![\mu v\ e]\!]//v\} && \text{(induct. hyp.)}\\
&= B([\![-]\!])(\mathsf{fp}\, v\ \varepsilon(e))\{[\![\mu v\ e]\!]//v\} && \text{(nat. of fp )}\\
&= B([\![-]\!])(\mathsf{fp}\, v\ \varepsilon(e)[\mu v\ e//v]) && \text{(Lemma 4.3.8)}\\
&= B([\![-]\!])\circ\varepsilon(\mu v\ e) && \square
\end{aligned}
$$

As a consequence of Theorem 4.3.9, we write $[\![-]\!]$ in place of $!_\varepsilon$ and simply refer to $[\![e]\!]$ as the semantics of $e$.

### 4.3.1 The Proof of Theorem 4.3.4

This section is dedicated to the proof of Theorem 4.3.4, which says that there is a unique $\Sigma_M$-algebra structure $\gamma\colon \Sigma_M Z \to Z$ satisfying the equations in (4.5).

$$
\begin{aligned}
\zeta(\gamma(v)) &= v \qquad & \zeta(\gamma(\sigma,t_1,\ldots,t_n)) &= \sigma(\zeta(t_1),\ldots,\zeta(t_n))\\
\zeta(\gamma(a,t)) &= (a,t) \qquad & \zeta(\gamma(\mu v,t)) &= \mathsf{fp}\, v\ \zeta(t)\{\gamma(\mu v,t)//v\}
\end{aligned}
\tag{4.5}
$$

We already saw in the text below Theorem 4.3.4 that we could break down the construction of such a $\gamma$ into $\gamma = [\gamma_{base}, \gamma_{rec}]$, where $\gamma_{base}\colon SZ + Var + Act\times Z \to Z$ and $\gamma_{rec}\colon \{\mu\}\times Var\times Z \to Z$. By Lambek's lemma [Lam68], $\zeta$ is invertible. This implies that the unique map $\gamma_{base}$ satisfying the first three equations of (4.5) is given by

$$
\begin{aligned}
\gamma_{base}(v) &= \zeta^{-1}(v) \qquad & \gamma_{base}(\sigma,t_1,\ldots,t_n) &= \zeta^{-1}(\sigma(\zeta(t_1),\ldots,\zeta(t_n)))\\
\gamma_{base}(a,t) &= \zeta^{-1}(a,t)
\end{aligned}
$$

Comparing this with (4.5), we are left with constructing the map $\gamma_{rec}$ and showing that it is the unique map satisfying the last of the four identities in (4.5).

**Lemma 4.3.10.** *There is at most one $r\in Z$ satisfying $\zeta(r) = \mathsf{fp}\, v\ \zeta(t)\{r//v\}$.*

*Proof.* We begin by showing that the relation

$$
R = \{(r\{s_1/v\}, r\{s_2/v\}) \mid \zeta(s_i) = \mathsf{fp}\, v\ \zeta(t)\{s_i//v\}, i\in\{1,2\}\}
$$

is a bisimulation on $(Z,\zeta)$. To this end, we define a coalgebra structure $\rho\colon R \to B_M R$

by induction on $\zeta(r)$ and show that the projection maps $\pi_1, \pi_2 \colon R \to Z$ are coalgebra homomorphisms. Let $\zeta(t) = p(v, \vec{u}, (a_1, t_1), \ldots, (a_n, t_n))$. If $\zeta(r) = v$, then $r\{s_i/v\} = s_i$ and we have

$$\zeta(s_i) = \mathsf{fp}\ v\ p(v, \vec{u}, (a_1, t_1\{s_i/v\}), \ldots, (a_n, t_n\{s_i/v\}))$$

Thus, we set

$$\begin{aligned}&\rho((r\{s_1/v\}, r\{s_2/v\}))\\ &= \mathsf{fp}\ v\ p(v, \vec{u}, (a_1, (t_1\{s_1/v\}, t_1\{s_2/v\})), \ldots, (a_n, (t_n\{s_1/v\}, t_n\{s_2/v\})))\end{aligned}$$

in this case, and deduce

$$\begin{aligned}&B_M(\pi_i) \circ \rho((r\{s_1/v\}, r\{s_2/v\}))\\ &= B_M(\pi_i)(\mathsf{fp}\ v\ p(v, \vec{u}, (a_1, (t_1\{s_1/v\}, t_1\{s_2/v\})), \ldots, (a_n, (t_n\{s_1/v\}, t_n\{s_2/v\})))) && \text{(def. } \rho\text{)}\\ &= \mathsf{fp}\ v\ p(v, \vec{u}, (a_1, t_1\{s_i/v\}), \ldots, (a_n, t_n\{s_i/v\})) && \text{(def. } B_M\text{)}\\ &= \mathsf{fp}\ v\ \zeta(t)\{s_i//v\}\\ &= \zeta(s_i) && \text{(assm. } s_i\text{)}\\ &= \zeta(r\{s_i/v\}) && (\zeta(r) = v)\\ &= \zeta \circ \pi_i((r\{s_1/v\}, r\{s_2/v\}))\end{aligned}$$

If $\zeta(r) = u \neq v$, then we set $\rho((r\{s_1/v\}, r\{s_2/v\})) = u$. If $\zeta(r) = \sigma(\zeta(r_1), \ldots, \zeta(r_n))$ and $B_M(\pi_i) \circ \rho((r_1\{s_1/v\}, r_1\{s_2/v\})) = \zeta(r_1\{s_i/v\})$, we set

$$\rho((r\{s_1/v\}, r\{s_2/v\})) = \sigma(\rho((r_1\{s_1/v\}, r_1\{s_2/v\})), \ldots, \rho((r_n\{s_1/v\}, r_n\{s_2/v\})))$$

and deduce

$$\begin{aligned}&B_M(\pi_i) \circ \rho((r\{s_1/v\}, r\{s_2/v\}))\\ &= B_M(\pi_i)\big(\sigma(\rho((r_1\{s_1/v\}, r_1\{s_2/v\})), \ldots, \rho((r_n\{s_1/v\}, r_n\{s_2/v\})))\big) && \text{(def. } \rho\text{)}\\ &= \sigma(B_M(\pi_i) \circ \rho((r_1\{s_1/v\}, r_1\{s_2/v\})), \ldots, B_M(\pi_i) \circ \rho((r_n\{s_1/v\}, r_n\{s_2/v\})))\\ & && (B_M(\pi_i) \text{ an alg. hom.})\\ &= \sigma(\zeta(r_1\{s_i/v\}), \ldots, \zeta(r_n\{s_i/v\})) && \text{(ind. hyp.)}\end{aligned}$$

$$= \zeta(r\{s_i/v\}) \qquad \text{(def. } \{-/v\}\text{)}$$
$$= \zeta \circ \pi_i((r\{s_1/v\}, r\{s_2/v\}))$$

Thus, $B_M(\pi_i) \circ \rho = \zeta \circ \pi_i$, so $R$ is a bisimulation. By finality of $(Z,\zeta)$, $R \subseteq \Delta_Z$, so $r\{s_1/v\} = r\{s_2/v\}$ for any $(r\{s_1/v\}, r\{s_2/v\}) \in R$. In particular, if $r = \zeta^{-1}(v)$, then $s_1 = s_2$. Hence, there is at most one $r$ such that $\zeta(r) = \mathsf{fp}\, v\ \zeta(t)\{r//v\}$. □

We are now left with the task of showing that a behaviour $s$ satisfying $\zeta(s) = \mathsf{fp}\, v\ \zeta(t)\{s//v\}$ exists. It clarifies the situation greatly to start by describing an analogous operator on a different final coalgebra, namely the final coalgebra for the functor $B_{S^*} = S^*(Var + Act \times \mathrm{Id})$. We start by equipping this final coalgebra with a complete metric, which will allow us to obtain recursion in $v$ as a certain limit.

Up to isomorphism, the final $S^*(Var + Act \times \mathrm{Id})$-coalgebra consists of the set $\Gamma$ of finite and infinite ordered trees with a particular decoration[7]: Intuitively, leaves of the tree are labelled either with closed $S$-terms or variables from $Var$, and each interior node with a list of $m$ children is labelled by a term $p(\vec{v}, (a_1, 1), \ldots, (a_m, m)) \in S^*(Var + Act \times \mathbb{N})$. The coalgebra structure $\beta : \Gamma \to B_{S^*}\Gamma$ on these trees assigns a tree $t$ with root node labelled $p(v_1, \ldots, v_n, (a_1, 1), \ldots, (a_m, m))$ to the term $\beta(t) = p(v_1, \ldots, v_n, (a_1, t_1), \ldots, (a_m, t_m))$, where $t_i$ is the $i$th child of $t$. All of this can be captured formally as follows.

**Definition 4.3.11.** Define $\mathrm{step}(p(\vec{v}, (a_1, x_1), \ldots, (a_n, x_n))) = \{x_1, \ldots, x_n\}$ for any $p \in B_{S^*}X$. A *coterm* is a partial function $t : \mathbb{N}^* \rightharpoonup B_{S^*}\mathbb{N}$ such that for any $w \in \mathbb{N}^*$,

1. There is an $n \in \mathbb{N}$ such that $\mathrm{step}(t(w)) = \{1, \ldots, n\}$.[8]
2. $\varepsilon \in \mathrm{dom}(t)$, where $\varepsilon$ is the empty word.
3. If $w \in \mathrm{dom}(t)$ and $n \in \mathrm{step}(t(w))$, then $wn \in \mathrm{dom}(t)$.
4. If $wn \in \mathrm{dom}(t)$, then $w \in \mathrm{dom}(t)$.

The set of all coterms is denoted $\Gamma$. The coalgebra structure $\beta : \Gamma \to B_{S^*}\Gamma$ is defined $\beta(t) = p(\vec{v}, (a_1, t_1), \ldots, (a_n, t_n))$ when $t(\varepsilon) = p(\vec{v}, (a_1, 1), \ldots, (a_n, n))$ and

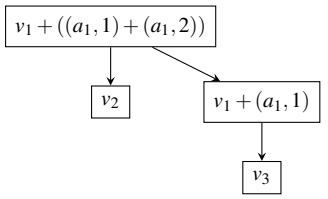
$$\mathrm{dom}(t_i) = \{w \in \mathbb{N}^* \mid iw \in \mathrm{dom}(t)\} \qquad t_i = \lambda w.t(iw)$$

[7] We are essentially going to follow [Acz+03] for the coalgebraic perspective. Explicit constructions on trees are inspired by [Cou83]

[8] In case $n = 0$, $\mathrm{step}(t(w)) = \emptyset$.

for $i \in \{1,\dots,n\}$. That is, $\beta(t) = p(\vec{v},(a_1,\lambda w.t(1w)),\dots,(a_n,\lambda w.t(nw)))$.

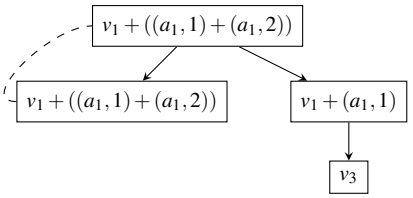
*Example* 4.3.12. For $S = \{+\} \times \mathrm{Id}^2$, the figure below is the coterm $t \in \Gamma$ such that $t(\varepsilon) = v_1 + ((a_1,1)+(a_1,2))$, $t(1) = v_2$, $t(2) = v_1 + (a_1,1)$, and $t(2\ 1) = v_3$.




$$\beta(t) = v_1 + ((a_1, \boxed{v_2}\ ) + (a_1, \boxed{v_1 + (a_1,1)\ \atop \downarrow\ v_3}\ )) \tag{4.6}$$

Intuitively, the tree above corresponds to the process term $v_1 + (a_1 v_2 + a_1(v_1 + a_1 v_3))$.

For an infinite example, consider the tree below.




$$\beta(t) = v_1 + ((a_1,t) + (a_1, \boxed{v_1 + (a_1,1)\ \atop \downarrow\ v_3}\ ))$$

Above, the dashed lines indicate that two subtrees are isomorphic. Intuitively, this coterm corresponds to $\mu v_2\ (v_1 + (a_1 v_2 + a_1(v_1 + a_1 v_3)))$.

**Lemma 4.3.13.** *The structure* $(\Gamma,\beta)$ *is a final* $B_{S^*}$*-coalgebra.*

Given a coalgebra $\delta\colon X \to B_{S^*}X$, the coterm $!_\delta(x)$ corresponding to a state $x \in X$ is given by unrolling. That is, if $\delta(x) = p(\vec{u},(a_1,x_1),\dots,(a_n,x_n))$, then $!_\delta(x)$ is the coterm whose root is labelled by $p(\vec{u},(a_1,1),\dots,(a_n,n))$, and whose children are the coterms $!_\delta(x_1),\dots,!_\delta(x_n)$.

*Proof.* Let $(X,\delta)$ be a $B_{S^*}$-coalgebra. Following the intuition above, we define $!_\delta\colon X \to \Gamma$ formally as follows. Let $x \in X$ and suppose $\delta(x) = p(\vec{u},(a_1,x_1),\dots,(a_n,x_n))$. We inductively define $!_\delta(x)(\varepsilon) = p(\vec{v},(a_1,1),\dots,(a_n,n))$ for the empty word, and for $i \in \{1,\dots,n\}$ and $w \in \mathrm{dom}(!_\delta(x_i))$, $!_\delta(x)(iw) = !_\delta(x_i)(w)$. The map $!_\delta$ is a coalgebra homomorphism because

$$\begin{aligned} B_{S^*}(!_\delta) \circ \delta(x) &= B_{S^*}(!_\delta)(p(\vec{v},(a_1,x_1),\dots,(a_n,x_n))) \\ &= p(\vec{v},(a_1,!_\delta(x_1)),\dots,(a_n,!_\delta(x_n))) \\ &= p(\vec{v},(a_1,\lambda w.!_\delta(x)(1w)),\dots,(a_n,\lambda w.!_\delta(x)(nw))) \end{aligned}$$

$$= \beta \circ !_\delta(x)$$

For uniqueness, suppose $h\colon (X,\delta) \to (\Gamma,\beta)$ is any other coalgebra homomorphism. We show by induction on $w \in \mathbb{N}^*$ that $w \in \mathrm{dom}(h(x))$ iff $w \in \mathrm{dom}(!_\delta(x))$ and that $h(x)(w) = !_\delta(x)(w)$. For $\varepsilon$, this amounts to observing that since $\delta(x) = p(\vec{v},(a_1,x_1),\ldots,(a_n,x_n))$, we must have

$$h(x)(\varepsilon) = p(\vec{v},(a_1,1),\ldots,(a_n,n)) = !_\delta(x)(\varepsilon)$$

For the inductive step, assume that $w \in \mathrm{dom}(h(y))$ iff $w \in \mathrm{dom}(!_\delta(y))$ and assume that $h(y)(w) = !_\delta(y)(w)$ for all $y \in X$. Let $i \in \{1,\ldots,n\}$, and observe that $iw \in \mathrm{dom}(h(x))$ and $iw \in \mathrm{dom}(!_\delta(x))$, and that for no other $i \in \mathbb{N}$ we find either $iw \in \mathrm{dom}(h(x))$ or $iw \in \mathrm{dom}(!_\delta(x))$. Thus, to finish the proof, simply observe that the induction hypothesis now tells us $h(x)(iw) = h(x_i)(w) = !_\delta(x_i)(w) = !_\delta(x)(iw)$. □

**The construction of $\mu v\, t$**

Recall that our primary goal in this section is to formally construct the behaviour $\mu v\, t$. We begin with a construction of an analogous operator $\mu v$, of the same name, on coterms. If $t$ is a coterm, then the coterm $\mu v\, t$ can be obtained as a *limit* with respect to a certain metric $d$ on coterms. Our next step is to define this metric $d$.

Write $\mathbb{N}^{\leq n}$ for the set of words $w \in \mathbb{N}^*$ such that the length $|w|$ of $w$ is at most $n$. We define the *depth-n restriction* of a coterm $t \in \Gamma$ to be $t|_n$, where $\mathrm{dom}(t|_n) = \mathrm{dom}(t) \cap \mathbb{N}^{\leq n}$ and $t|_n(w) = t(w)$ for any $w \in \mathrm{dom}(t|_n)$.

**Definition 4.3.14.** The *coterm metric* is the map $d\colon \Gamma \times \Gamma \to [0,1]$ defined as follows: For $s,t \in \Gamma$,

$$d(s,t) = \inf\{2^{-(n+1)} \mid t|_n = s|_n\}$$

if there is an $n$ such that $t|_n = s|_n$, and otherwise $d(s,t) = 1$.

*Remark* 4.3.15. The metric $d$ defined above has been obtained by instantiating a more general construction of a complete metric on a final coalgebra due to Barr [Bar93] (see also [Adá03] and the much earlier study of complete metric spaces of trees [AN80]). Barr constructs an analogue of the metric $d$ on the final coalgebra of any bicontinuous functor on **Set** using its *final sequence*, an $\omega^{op}$-sequence with a final coalgebra as a limit. The functor $B_{S^*}$ is bicontinuous, so Barr's general

construction applies. We spell out the metric $d$ and prove the required properties of $d$ concretely to make the chapter as self-contained as possible.

**Lemma 4.3.16.** *The coterm metric satisfies the following property: For any $s,t \in \Gamma$,*

$$d(s,t) = \frac{1}{2}\max\{d(s_j,t_j) \mid j \leq m\}$$

*if $\beta(s) = p(\vec{u},(a_1,s_1),\ldots,(a_n,s_m))$ and $\beta(t) = p(\vec{u},(a_1,t_1),\ldots,(a_n,t_m))$, and otherwise $d(s,t) = 1$. Furthermore, the pair $(\Gamma,d)$ is a complete metric space.*

*Proof.* If $\beta(s) = p(\vec{v},(a_1,s_1),\ldots,(a_n,s_n))$ and $\beta(t) = p(\vec{v},(a_1,t_1),\ldots,(a_n,t_n))$, then $t|_{n+1} = s|_{n+1}$ iff for any $i \leq m$, $t_i|_n = s_i|_n$. Thus, $d(s,t) = \frac{1}{2}\max\{d(s_j,t_j) \mid j \leq m\}$.

The space $(\Gamma,d)$ is complete because in any Cauchy sequence of coterms $\{t_i\}_{i\in\mathbb{N}}$, there is a subsequence $\{t_{k_i}\}_{i\in\mathbb{N}}$ such that for any $i \in \mathbb{N}$, the coterms $t_{k_i}$ and $t_{k_{i+1}}$ must agree on all words of length at most $i$. This subsequence is obtained by observing that since $\{t_i\}_{i\in\mathbb{N}}$ is Cauchy, given any $i \in \mathbb{N}$, there is a $k_i \in \mathbb{N}$ such that for any $j_1, j_2 \geq k_i$, $d(t_{j_1},t_{j_2}) \leq 2^{-(i+1)}$. For any such $j_1, j_2$, $t_{j_1}|_i = t_{j_2}|_i$. The limit of $\{t_i\}_{i\in\mathbb{N}}$ is the unique coterm $t$ such that $t|_i = t_{k_i}|_i$ for all $i \in \mathbb{N}$. □

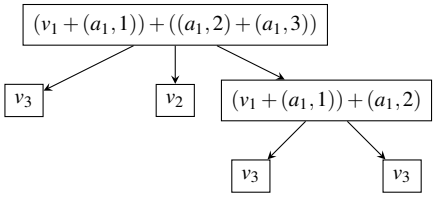
Given an $s \in \Gamma$ and $v \in \mathit{Var}$, we use finality of $(\Gamma,\beta)$ to corecursively define the substitution operator $(-)[s/v]\colon \Gamma \to \Gamma$ as follows: Given $\beta(t) = p(v,\vec{u},(a_1,t_1),\ldots,(a_n,t_n))$,

$$\beta(t[s/v]) = p(\beta(s),\vec{u},(a_1,t_1[s/v]),\ldots,(a_n,t_n[s/v])) \tag{4.7}$$

Note the similarity to the definition of behavioural substitution (Definition 4.3.2). Also note that we clearly have $t[v/v] = t$.

The substitution $t[s/v]$ amounts to replacing each $v$ appearing in some label of a node in $t$ with the root label of $s$ (and then renumbering), and then inserting the list of children of $s$ in the appropriate places in the list of children of $t$.

*Example* 4.3.17. If $t$ is the coterm in (4.6) and $s$ is the third child of $t$, then $t[s/v_1]$ is the coterm below:



The label of the root of $t$ is $v_1 + ((a_1,1) + (a_1,2))$. Replacing $v_1$ with the label of the root of $s$ gives $(v_1 + (a_1,1)) + ((a_1,1) + (a_2,2))$. Renumbering so that the step indices correspond to the children gives $(v_1 + (a_1,1)) + ((a_1,2) + (a_2,3))$, and appending the list of children of $s$ to the beginning of the list of children of $t$ gives the root node above and its children. Repeating this for the third child gives the coterm above.

For $t, s_1, s_2 \in \Gamma$, it is straightforward to see that $d(t[s_1/v], t[s_2/v]) \leq d(s_1, s_2)$. Indeed, there is a minimal length $n$ for which the variable $v$ appears free in $t(w)$ with $|w| = n$. If $d(s_1, s_2) = 2^{-k}$, then $d(t[s_1/v], t[s_2/v]) \leq 2^{-(n+k)}$, since $t[s_1/v](u) = t(u) = t[s_2/v](u)$ for every word $u$ with $|u| < n$. This means that $t[-/v]$ is *nonexpanding*. The next lemma tells us that substituting into certain coterms is a *contraction* by $\frac{1}{2}$.

**Lemma 4.3.18.** *Let $t \in \Gamma$, and assume that $v$ is not free in $\beta(t)$. Then for any $s_1, s_2 \in \Gamma$,*

$$d(t[s_1/v], t[s_2/v]) \leq \frac{1}{2} d(s_1, s_2)$$

*Proof.* Since $v$ is not free in $\beta(t)$, for $i$ either 1 or 2,

$$\beta(t[s_i/v]) = p(\vec{u}, (a_1, t_1[s_i/v]), \ldots, (a_m, t_m[s_i/v]))$$

Therefore,

$$d(t[s_1/v], t[s_2/v]) = \frac{1}{2} \max\{d(t_j[s_1/v], t_j[s_2/v]) \mid j \leq m\} \leq \frac{1}{2} d(s_1, s_2)$$

because $t[-/v]$ is nonexpanding. □

The following lemma allows us to define recursion in the variable $v$ on coterms.

**Lemma 4.3.19.** *Let $t \in \Gamma$ such that $v$ is not free in $\beta(t)$. Then the function $t[-/v]$ has a unique fixed-point.*

*Proof.* Recall that nonexpanding maps are also continuous. Given $t \in \Gamma$ such that $v$ is not free in $\beta(t)$, a unique fixed-point of the map $t[-/v]$ can be obtained as follows.[9] Since $v$ is not free in $\beta(t)$, clearly $d(v,t) = 1$ (here, $v$ denotes the unique coterm with $\beta(v) = v$). By Lemma 4.3.18, $d(t, t[t/v]) = d(t[v/v], t[t/v]) \leq \frac{1}{2}$. Let $t_0 = v$ and

[9] Note that we could also have used the *Banach fixed-point theorem* [Ban22].

$t_{n+1} = t[t_n/v]$. By induction, for $n > 0$, we have

$$d(t_n, t_{n+1}) = d(t[t_{n-1}/v], t[t_n/v]) \leq \frac{1}{2} d(t_{n-1}, t_n)$$

It follows that $\{t_i\}_{i \in \mathbb{N}}$ is a Cauchy sequence and therefore has a limit, call it $t'$. By continuity of $t[-/v]$, $t' = t[t'/v]$. Moreover, if $s$ also satisfies $s = t[s/v]$, we must have

$$d(t', s) = d(t[t'/v], t[s/v]) \leq \frac{1}{2} d(t', s)$$

so it must be the case that $s = t'$. That is, $t'$ is the unique fixed-point of $t[-/v]$. □

To make the definition of $\mu v\, t$ precise, write $\mathsf{fp}\, v\, t$ for the unique coterm $s$ such that $\beta(s) = \mathsf{fp}\, v\, \beta(t)$. The point is that $v$ is never free in $\mathsf{fp}\, v\, \beta(t)$.

**Definition 4.3.20.** Given $t \in \Gamma$, we write $F_v(t)$ for the unique fixed-point of $\mathsf{fp}\, v\, t[-/v]$.

We finish the proof of Theorem 4.3.4 by showing that $F_v(t)$ is the key ingredient in the definition of $\gamma_{rec}$, i.e., the recursion operators $\gamma(\mu v, -)$ on $Z$.

Mapping every $S$-term into its EQ-equivalence class is a surjective natural transformation $[-]_{\mathsf{EQ}} \colon S^* \Rightarrow M$, and from it we obtain a surjective natural transformation $S^*(Var + Act \times \mathrm{Id}) \Rightarrow M(Var + Act \times \mathrm{Id})$ (of the same name). Since $(Z, \zeta)$ is the final $B_M$-coalgebra, we obtain a unique coalgebra homomorphism $[-]^*_{\mathsf{EQ}} \colon \Gamma \to Z$ in the diagram below.

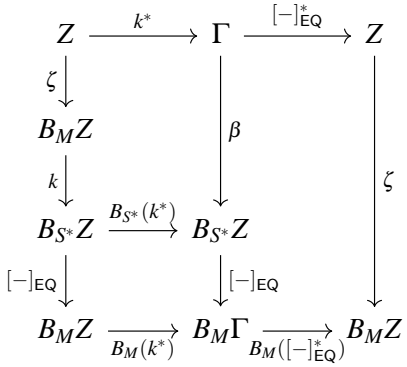
$$\begin{array}{ccc} \Gamma & \xrightarrow{\quad [-]^*_{\mathsf{EQ}} \quad} & Z \\ {\scriptstyle\beta}\downarrow & & \\ S^*(Var + Act \times \Gamma) & & \downarrow{\scriptstyle\zeta} \\ {\scriptstyle[-]_{\mathsf{EQ}}}\downarrow & & \\ M(Var + Act \times \Gamma) & \dashrightarrow & M(Var + Act \times Z) \end{array} \tag{4.8}$$

The map $[-]^*_{\mathsf{EQ}}$ allows for a presentation of process behaviours by coterms, i.e., the map $[-]^*_{\mathsf{EQ}}$ is surjective.

**Lemma 4.3.21.** *For any $t \in Z$, there is an $s \in \Gamma$ such that $t = [s]^*_{\mathsf{EQ}}$.*

*Proof.* By surjectivity of $[-]_{\mathsf{EQ}} \colon S^*(Var + Act \times Z) \to M(Var + Act \times Z)$, there is a map $k \colon M(Var + Act \times Z) \to S^*(Var + Act \times Z)$ such that $[-]_{\mathsf{EQ}} \circ k = \mathrm{id}$. We obtain $k^*$ in the

diagram below from finality of $(Z,\zeta)$.



(4.9)

The rectangles on the right and in the top left commute by definition. The bottom left square commutes by naturality of $[-]_{\mathsf{EQ}}$. Thus, $[-]^*_{\mathsf{EQ}} \circ k^*$ is a coalgebra homomorphism. By finality of $(Z,\zeta)$, $[-]^*_{\mathsf{EQ}} \circ k^* = \mathrm{id}$. □

We would now like to set $\gamma(\mu v,t) = [F_v(s)]^*_{\mathsf{EQ}}$, but we need to check that $[F_v(s)]^*_{\mathsf{EQ}}$ satisfies the right identity and determines a well-defined function $\gamma(\mu v,-)$. Both of these properties follow from Lemma 4.3.22 below.

**Lemma 4.3.22.** *Let $t \in \Gamma$. Then $[F_v(t)]^*_{\mathsf{EQ}}$ satisfies the fourth identity in (4.5) from Theorem 4.3.4, i.e., it is the unique solution to the equation $\zeta(r) = \mathsf{fp}\, v\, \zeta([t]^*_{\mathsf{EQ}})\{r//v\}$ in the indeterminate $r$.*

The proof of Lemma 4.3.22 requires the following lemma, which tells us that semantic subtitutions can be calculated using coterm substitution.

**Lemma 4.3.23.** *For any $s,t \in \Gamma$, $[s[t/v]]^*_{\mathsf{EQ}} = [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}$.*

*Proof.* We are going to show that the relation

$$R = \{([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}) \mid s,t \in \Gamma\} \cup \Delta_Z$$

is a bisimulation in $(Z,\zeta)$. We construct $\rho : R \to B_M R$ by induction on $\beta(s)$. For the first base case, if $\beta(s) = v$, then

$$\begin{aligned}
\zeta([s[t/v]]^*_{\mathsf{EQ}}) &= B_M([-]^*_{\mathsf{EQ}}) \circ [-]_{\mathsf{EQ}} \circ \beta(s[t/v]) && ([-]^*_{\mathsf{EQ}} \text{ a homom.}) \\
&= B_M([-]^*_{\mathsf{EQ}}) \circ [-]_{\mathsf{EQ}} \circ \beta(t) && (\text{def. } [-/v]) \\
&= \zeta([t]^*_{\mathsf{EQ}}) && ([-]^*_{\mathsf{EQ}} \text{ a homom.})
\end{aligned}$$

$$= \zeta([s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}) \qquad (\zeta([s]^*_{\mathsf{EQ}}) = v)$$

We let $\rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))$ agree with the diagonal bisimulation in this case. Similarly for $\beta(s) = u \neq v$. For the second base case, if $\beta(s) = (a, s')$, then

$$\begin{aligned}
\zeta([s[t/v]]^*_{\mathsf{EQ}}) &= B_M([-]^*_{\mathsf{EQ}}) \circ [-]_{\mathsf{EQ}} \circ \beta(s[t/v]) && ([-]^*_{\mathsf{EQ}} \text{ a homom.}) \\
&= B_M([-]^*_{\mathsf{EQ}})([(a, s'[t/v])]_{\mathsf{EQ}}) && (\text{def. } [-/v]) \\
&= [(a, [s'[t/v]]^*_{\mathsf{EQ}})]_{\mathsf{EQ}} \\
\zeta([s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}) &= [(a, [s']^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})]_{\mathsf{EQ}} && (\zeta([s]^*_{\mathsf{EQ}}) = (a, [s']^*_{\mathsf{EQ}}))
\end{aligned}$$

Thus, we set

$$\rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) = [(a, ([s'[t/v]]^*_{\mathsf{EQ}}, [s']^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))]_{\mathsf{EQ}}$$

in this case. We see that

$$\begin{aligned}
&B_M(\pi_1) \circ \rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) \\
&= B_M(\pi_1)([(a, ([s'[t/v]]^*_{\mathsf{EQ}}, [s']^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))]_{\mathsf{EQ}}) \\
&= [(a, [s'[t/v]]^*_{\mathsf{EQ}})]_{\mathsf{EQ}} \\
&= \zeta([s[t/v]]^*_{\mathsf{EQ}}) && (\beta(s) = (a, s')) \\
&B_M(\pi_2) \circ \rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) \\
&= B_M(\pi_2)([(a, ([s'[t/v]]^*_{\mathsf{EQ}}, [s']^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))]_{\mathsf{EQ}}) \\
&= [(a, [s']^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})]_{\mathsf{EQ}} \\
&= \zeta([s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}) && (\zeta([s]^*_{\mathsf{EQ}}) = [(a, [s']^*_{\mathsf{EQ}})]_{\mathsf{EQ}})
\end{aligned}$$

as desired.

For the inductive step, assume $\zeta(s) = \sigma(\zeta(s_1), \ldots, \zeta(s_n))$ and

$$\begin{aligned}
B_M(\pi_1) \circ \rho(([s_i[t/v]]^*_{\mathsf{EQ}}, [s_i]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) &= \zeta([s_i[t/v]]^*_{\mathsf{EQ}}) \\
B_M(\pi_2) \circ \rho(([s_i[t/v]]^*_{\mathsf{EQ}}, [s_i]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) &= \zeta([s_i]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})
\end{aligned}$$

for $i \leq n$. In this case, we set

$$\begin{aligned}
&\rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) \\
&= \sigma(\rho(([s_1[t/v]]^*_{\mathsf{EQ}}, [s_1]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})), \ldots, \rho(([s_n[t/v]]^*_{\mathsf{EQ}}, [s_n]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})))
\end{aligned}$$

Since $B_M(\pi_j)$ is an $S$-algebra homomorphism, by the induction hypothesis we have

$$\begin{aligned}
&B_M(\pi_j) \circ \rho(([s[t/v]]^*_{\mathsf{EQ}}, [s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) \\
&= B_M(\pi_j)(\sigma(\rho(([s_1[t/v]]^*_{\mathsf{EQ}}, [s_1]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})), \\
&\qquad\qquad \ldots, \rho(([s_n[t/v]]^*_{\mathsf{EQ}}, [s_n]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})))) \\
&= \sigma(B_M(\pi_j) \circ \rho(([s_1[t/v]]^*_{\mathsf{EQ}}, [s_1]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})), \\
&\qquad\qquad \ldots, B_M(\pi_j) \circ \rho(([s_n[t/v]]^*_{\mathsf{EQ}}, [s_n]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))) \\
&= \sigma(\zeta \circ \pi_j(([s_1[t/v]]^*_{\mathsf{EQ}}, [s_1]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})), \\
&\qquad\qquad \ldots, \zeta \circ \pi_j(([s_n[t/v]]^*_{\mathsf{EQ}}, [s_n]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}))) && \text{(ind. hyp.)} \\
&= \begin{cases} \sigma(\zeta([s_1[t/v]]^*_{\mathsf{EQ}}), \ldots, \zeta([s_n[t/v]]^*_{\mathsf{EQ}})) & j = 1 \\ \sigma(\zeta([s_1]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}), \ldots, \zeta([s_n]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\})) & j = 2 \end{cases} \\
&= \begin{cases} \zeta([s[t/v]]^*_{\mathsf{EQ}}) & j = 1 \\ \zeta([s]^*_{\mathsf{EQ}}\{[t]^*_{\mathsf{EQ}}/v\}) & j = 2 \end{cases} && \text{(see prev. calculation)}
\end{aligned}$$

as desired. It follows that $R$ is a bisimulation. □

We will also need the following observation: That $[-]^*_{\mathsf{EQ}}$ preserves $\mathsf{fp}\, v$. This can be shown as follows:

$$\begin{aligned}
\zeta([\mathsf{fp}\, v\, t]^*_{\mathsf{EQ}}) &= B_M([-]^*_{\mathsf{EQ}})([\beta(\mathsf{fp}\, v\, t)]_{\mathsf{EQ}}) && ([-]^*_{\mathsf{EQ}} \text{ a homom.}) \\
&= B_M([-]^*_{\mathsf{EQ}})([\mathsf{fp}\, v\, \beta(t)]_{\mathsf{EQ}}) && (\text{def. } \mathsf{fp}\, v) \\
&= B_M([-]^*_{\mathsf{EQ}})(\mathsf{fp}\, v\, [\beta(t)]_{\mathsf{EQ}}) && (\text{def. branching theory}) \\
&= \mathsf{fp}\, v\, B_M([-]^*_{\mathsf{EQ}})([\beta(t)]_{\mathsf{EQ}}) && (\mathsf{fp} \text{ nat.}) \\
&= \mathsf{fp}\, v\, \zeta([t]^*_{\mathsf{EQ}}) && ([-]^*_{\mathsf{EQ}} \text{ a homom.})
\end{aligned}$$

With Lemmas 4.3.10 and 4.3.23 and the above observation in hand, we are finally ready to prove Lemma 4.3.22, which establishes $[F_v(t)]^*_{\mathsf{EQ}}$ as the unique solution to

the equation $\zeta(r) = \mathsf{fp}\, v\ \zeta(t)\{r//v\}$ in the indeterminate $r$.

*Proof of Lemma 4.3.22.* To see that $[F_v(t)]^*_{\mathsf{EQ}}$ satisfies the desired equation, we calculate

$$\begin{aligned}
\zeta([F_v(t)]^*_{\mathsf{EQ}}) &= \zeta([\mathsf{fp}\, v\ t[F_v(t)/v]]^*_{\mathsf{EQ}}) && \text{(def. } F_v\text{)}\\
&= \zeta([\mathsf{fp}\, v\ t]^*_{\mathsf{EQ}}\{[F_v(t)]^*_{\mathsf{EQ}}/v\}) && \text{(Lemma 4.3.23)}\\
&= \zeta(\mathsf{fp}\, v\ [t]^*_{\mathsf{EQ}}\{[F_v(t)]^*_{\mathsf{EQ}}/v\}) && \text{(see above)}\\
&= \zeta(\mathsf{fp}\, v\ [t]^*_{\mathsf{EQ}})\{[F_v(t)]^*_{\mathsf{EQ}}//v\} && (v \text{ not free in } \zeta(\mathsf{fp}\, v\ -))\\
&= \mathsf{fp}\, v\ \zeta([t]^*_{\mathsf{EQ}})\{[F_v(t)]^*_{\mathsf{EQ}}//v\} && \text{(def. fp } v\text{)}
\end{aligned}$$

Thus, $\zeta([F_v(t)]^*_{\mathsf{EQ}}) = \mathsf{fp}\, v\ \zeta([t]^*_{\mathsf{EQ}})\{[F_v(t)]^*_{\mathsf{EQ}}//v\}$. By Lemma 4.3.10, $[F_v(t)]^*_{\mathsf{EQ}}$ is the unique behaviour satisfying this equation. □

At last, Lemma 4.3.22 tells us that for any $t \in Z$ and $v \in \mathit{Var}$, we can define $\gamma_{rec}(\mu v, t) = [F_v(s)]^*_{\mathsf{EQ}}$ for some $s \in \Gamma$ such that $[s]^*_{\mathsf{EQ}} = t$, and furthermore that this does does not depend on the representative $s$ chosen.

## 4.3.2 The Proof of Lemma 4.3.7

The purpose of this section is to prove Lemma 4.3.7, that for any $e, g \in \mathit{Exp}$ such that no free variable of $g$ appears bound in $e$, we have $[\![e[g/v]]\!] = [\![e]\!]\{[\![g]\!]/v\}$ for any $v \in \mathit{Var}$. Several lemmas are needed, which we present here. The following theorem provides a simplified coinductive principle we will make use of in the proofs below.

**Theorem 4.3.24.** *If $h, k \colon Z \to Z$ satisfy properties (i)-(iii) below, then $h = k$.*

*(i) If $\zeta(t) = w$, then $\zeta(h(t)) = \zeta(k(t))$.*
*(ii) If $\zeta(t) = (a, r)$, then $\zeta(h(t)) = (a, h(r))$ and $\zeta(k(t)) = (a, k(r))$.*
*(iii) If $\zeta(t) = \sigma(\zeta(t_1), \ldots, \zeta(t_n))$, then*

$$\zeta(h(t)) = \sigma(\zeta(h(t_1)), \ldots, \zeta(h(t_n))) \quad \text{and} \quad \zeta(k(t)) = \sigma(\zeta(k(t_1)), \ldots, \zeta(k(t_n)))$$

*Proof.* The gist of the proof below is that (i)-(iii) guarantee that $h, k$ are coalgebra homomorphisms. Since there exactly one coalgebra homomorphism $(X, \delta) \to (Z, \zeta)$,

it must be that $h = k$. Formally, we are going to show that the relation

$$R = \{(h(t), k(t)) \mid t \in Z\}$$

is a bisimulation. That is, we define a $B_M$-coalgebra structure $\rho : R \to B_M R$ such that the projections $\pi_i : R \to Z$, $i = 1,2$, are coalgebra homomorphisms. By finality of $(Z, \zeta)$, this then implies that $\pi_1 = !_\rho = \pi_2$, or equivalently $R \subseteq \Delta_Z$, which establishes the identity we are hoping to prove.

Consider a $t \in Z$ and let $\zeta(t) = p(\vec{v}, (a_1, s_1), \ldots, (a_n, s_n)) \in M(Var + Act \times Z)$. We proceed by induction on $p(\vec{v}, (a_1, s_1), \ldots, (a_n, s_n))$ as follows:

- If $\zeta(t) = v$, then set $\rho(h(t), k(t)) = v$.
- If $\zeta(t) = (a, s)$, then set $\rho(h(t), k(t)) = (a, (h(s), k(s)))$.
- If $\zeta(t) = \sigma(\zeta(t_1), \ldots, \zeta(t_n))$, then set

$$\rho(h(t), k(t)) = \sigma(\rho(h(t_1), k(t_1)), \ldots, \rho(h(t_n), k(t_n)))$$

This results in the explicit form

$$\rho(h(t), h(t)) = p(\vec{v}, (a_1, (h(s_1), k(s_1))), \ldots, (a_1, (h(s_n), k(s_n))))$$

To see that $\pi_1, \pi_2$ are coalgebra homomorphisms,

$$\begin{aligned} B_M(\pi_1)(\rho(h(t), k(t))) &= M(\pi_1)(p(\vec{v}, (a_1, (h(s_1), k(s_1))), \ldots, (a_n, (h(s_n), k(s_n))))) \\ &= p(\vec{v}, (a_1, h(s_1)), \ldots, (a_n, h(s_n))) \\ &= \zeta \circ \pi_1(h(t), k(t)) \end{aligned}$$

The second to last inequality is due to (i)-(iii), which ensures that

$$\zeta(h(t)) = p(\vec{v}, (a_1, h(s_1)), \ldots, (a_n, h(s_n)))$$

Similarly for $\pi_2$. □

For the following lemma, we say that a variable $v$ is *absent* in a behaviour $t$ if for any other behaviour $s$, $t\{s/v\} = t$.

**Lemma 4.3.25.** *Let $u,v \in Var$ and $r,s,t \in Z$. If $v$ is absent in $t$ and $u \neq v$, then*

$$r\{s/v\}\{t/u\} = r\{t/u\}\{s\{t/u\}/v\}$$

*Proof.* We use Theorem 4.3.24, which requires us to verify (i)-(iii) for the maps $(-)\{s/v\}\{t/u\}$ and $(-)\{t/u\}\{s\{t/u\}/v\}$. Assume $u,v,w$ are distinct variables, and let $a \in Act$. There are several cases to consider.

(i) If $\zeta(r) = v$, then $\zeta(r\{s/v\}) = \zeta(s)$ and $\zeta(r\{t/u\}) = \zeta(r)$. This means that

$$\zeta(r\{s/v\}\{t/u\}) = \zeta(s\{t/u\}) = \zeta(r\{s\{t/u\}/v\}) = \zeta(r\{t/u\}\{s\{t/u\}/v\})$$

(i') If $\zeta(r) = u$, then $\zeta(r\{s/v\}) = \zeta(r)$ and $\zeta(r\{t/u\}) = \zeta(t)$. This means that

$$\zeta(r\{s/v\}\{t/u\}) = \zeta(r\{t/u\}) = \zeta(t) = \zeta(t\{s\{t/u\}/v\}) = \zeta(r\{t/u\}\{s\{t/u\}/v\})$$

(i") If $\zeta(r) = w \notin \{u,v\}$, then $\zeta(r\{s/v\}\{t/u\}) = \zeta(r) = \zeta(r\{t/u\}\{s\{t/u\}/v\})$.

(ii) If $\zeta(r) = (a,r')$, then $\zeta(r\{s/v\}) = (a,r'\{s/v\})$ and $\zeta(r\{t/u\}) = (a,r'\{t/u\})$. It follows that

$$\zeta(r\{s/v\}\{t/u\}) = (a,r'\{s/v\}\{t/u\})$$

and

$$\zeta(r\{t/u\}\{s\{t/u\}/v\}) = (a,r'\{t/u\}\{s\{t/u\}/v\})$$

(iii) Now let $\zeta(r) = \sigma(\zeta(r_1),\ldots,\zeta(r_n))$. By definition,

$$\zeta(r\{s/v\}\{t/u\}) = \sigma(\zeta(r_1\{s/v\}\{t/u\}),\ldots,\zeta(r_n\{s/v\}\{t/u\}))$$

and

$$\zeta(r\{t/u\}\{s\{t/u\}/v\}) = \sigma(\zeta(r_1\{t/u\}\{s\{t/u\}/v\}),\ldots,\zeta(r_n\{t/u\}\{s\{t/u\}/v\}))$$

as desired. □

**Lemma 4.3.26.** *For any $r,s,t \in Z$ and $v \in Var$, $r\{s/v\}\{t/v\} = r\{s\{t/v\}/v\}$.*

*Proof.* This also follows from Theorem 4.3.24, where this time $h = (-)\{s/v\}\{t/v\}$ and $k = (-)\{s\{t/v\}/v\}$. □

**Lemma 4.3.27.** *Let $v \in Var$ and $t,s \in Z$. If $v$ is absent in $t$, then $\zeta(t)\{s//v\} = \zeta(t)$.*

*Proof.* We proceed by induction on $\zeta(t) = p(\vec{v},(a_1,s_1),\ldots,(a_n,s_n)) \in M(Var + Act \times Z)$.

- In case $\zeta(t) = u \neq v$, we have $u\{s//v\} = u = \zeta(t)$.
- Now let $\zeta(t) = (a_1,s_1)$. Since $v$ is absent in $t$, $(a_1,s_1)\{s//v\} = (a_1,s_1\{s/v\}) = \zeta(t\{s/v\}) = \zeta(t)$.
- For the inductive step, let $\zeta(t) = \sigma(\zeta(t_1),\ldots,\zeta(t_n))$. By the induction hypothesis,

$$\zeta(t)\{s//v\} = \sigma(\zeta(t_1)\{s//v\},\ldots,\zeta(t_n)\{s//v\}) = \sigma(\zeta(t_1),\ldots,\zeta(t_n)) = \zeta(t) \qquad \square$$

**Lemma 4.3.28.** *If $v$ is absent in $t$, then $\zeta(t) = \mathsf{fp}\, v\ \zeta(t)$ and $\mu v\ t = t$.*

*Proof.* Suppose $\zeta(t) = p(v,\vec{u},(a_1,s_1),\ldots,(a_n,s_n))$ with $v \neq u_i$ for any index $i$. If $v$ is absent in $t$, then $t = t\{0/v\}$, so that

$$\begin{aligned}
\zeta(t) &= \zeta(t\{0/v\}) \\
&= p(0,\vec{u},(a_1,s_1\{0/v\}),\ldots,(a_n,s_n\{0/v\})) \\
&= \mathsf{fp}\, v\ p(0,\vec{u},(a_1,s_1\{0/v\}),\ldots,(a_n,s_n\{0/v\})) && \text{(no } v \text{ to reduce)} \\
&= \mathsf{fp}\, v\ \zeta(t\{0/v\}) \\
&= \mathsf{fp}\, v\ \zeta(t)
\end{aligned}$$

To see the latter statement,

$$\zeta(\mu v\ t) = \mathsf{fp}\, v\ \zeta(t)\{\mu v\ t//v\} = \zeta(t)\{\mu v\ t//v\} = \zeta(t)$$

by Lemma 4.3.27. $\square$

**Lemma 4.3.29.** *Let $u,v$ be distinct variables and $s,t \in Z$. If $v$ is absent in $s$, then*

$$(\mu v\ t)\{s/u\} = \mu v\ (t\{s/u\})$$

*Proof.* The behaviour $\mu v\ (t\{s/u\})$ is (by Lemma 4.3.10) the unique solution to the behavioural differential equation

$$\zeta(r) = \mathsf{fp}\, v\ \zeta(t\{s/u\})\{r//v\} \qquad (*)$$

in the indeterminate $r$. Thus, it suffices to see that $r = (\mu v\ t)\{s/u\}$ satisfies this equation. To this end, let $\zeta(t) = \sigma(\zeta(t_1), \ldots \zeta(t_n))$, so that

$$\zeta(\mu v\, t) = \mathsf{fp}\ v\ p(v, \vec{w}, (a_1, s_1\{\mu v\ t/v\}), \ldots, (a_n, s_n\{\mu v\ t/v\}))$$

We have

$$\begin{aligned}
&\zeta((\mu v\ t)\{s/u\}) \\
&= \mathsf{fp}\ v\ p(v, \vec{w}, (a_1, s_1\{\mu v\ t/v\}\{s/v\}), \ldots, (a_n, s_n\{\mu v\ t/v\}\{s/v\})) \\
&= \mathsf{fp}\ v\ p(v, \vec{w}, (a_1, s_1\{s/v\}\{(\mu v\ t)\{s/v\}/v\}), \ldots, (a_n, s_n\{s/v\}\{(\mu v\ t)\{s/v\}/v\})) \\
&\qquad\qquad \text{(Lemma 4.3.27)} \\
&= \mathsf{fp}\ v\ \zeta(t\{s/v\})\{(\mu v\ t)\{s/v\}//v\} \qquad \square
\end{aligned}$$

**Lemma 4.3.30.** *Let $v \in Var$ and $e \in Exp$. If $v$ is not free in $e$, then $v$ is absent in $[\![e]\!]$.*

*Proof.* We show that $\zeta([\![e]\!]\{s/v\}) = \zeta([\![e]\!])$ for all $s$ by induction on $e$. In the variable case, $e = u \neq v$. Here, $\zeta([\![u]\!]) = u$, so $\zeta([\![u]\!]\{s/v\}) = \zeta([\![u]\!])$.

Now suppose the result is true for $e$. Since $v$ is free in $ae$ if and only if $v$ is free in $e$, it must be the case that $v$ is not free in $e$. By the induction hypothesis, $\zeta([\![ae]\!]\{s/v\}) = (a, [\![e]\!]\{s/v\}) = (a, [\![e]\!]) = \zeta([\![ae]\!])$.

Next, assume the result for $e_1, \ldots, e_n$, and let $\sigma$ be an $S$-operation. Since $v$ is not free in $\sigma(e_1, \ldots, e_n)$ if and only if $v$ is not free in any of the $e_i$, and since $\zeta([\![\sigma(e_1, \ldots, e_n)]\!]) = \sigma(\zeta([\![e_1]\!]), \ldots, \zeta([\![e_n]\!]))$, we have

$$\begin{aligned}
\zeta([\![\sigma(e_1, \ldots, e_n)]\!]\{s/v\}) &= \sigma(\zeta([\![e_1]\!]\{s/v\}), \ldots, \zeta([\![e_n]\!]\{s/v\})) && \text{(def. } \{-/v\}, [\![-]\!]) \\
&= \sigma(\zeta([\![e_1]\!]), \ldots, \zeta([\![e_n]\!])) && \text{(induct. hyp.)} \\
&= \zeta([\![\sigma(e_1, \ldots, e_n)]\!])
\end{aligned}$$

Now assume the result for $e$ and let $u \in Var$. In this case, we consider the expression $\mu u\ e$ and the following two subcases.

- If $u = v$, then $v$ is not free in $\mu u\ e$ and therefore satisfies the hypotheses of the

lemma. In this case, let $\zeta(\llbracket e \rrbracket) = p(v, \vec{u}, (a_1, s_1), \ldots, (a_n, s_n))$. Then

$$\begin{aligned}
&\zeta(\llbracket \mu v\ e \rrbracket \{s/v\}) \\
&= \mathsf{fp}\, v\ p(v, \vec{u}, (a_1, s_1\{\llbracket \mu v\ e \rrbracket /v\}\{s/v\}), \ldots, (a_n, s_n\{\llbracket \mu v\ e \rrbracket /v\}\{s/v\})) \\
&= \mathsf{fp}\, v\ p(v, \vec{u}, (a_1, s_1\{\llbracket \mu v\ e \rrbracket \{s/v\}/v\}), \ldots, (a_n, s_n\{\llbracket \mu v\ e \rrbracket \{s/v\}/v\})) \\
&= \mathsf{fp}\, v\ \zeta(\llbracket e \rrbracket)\{\llbracket \mu v\ e \rrbracket \{s/v\}/v\}
\end{aligned}$$

Now, $\llbracket \mu u\ e \rrbracket$ is the unique solution to the behavioural differential equation $\zeta(r) = \zeta(\llbracket e \rrbracket)\{r//u\}$ in the indeterminate $r$, and $r = \llbracket \mu v\ e \rrbracket \{s/t\}$ satisfies this equation. It must be that $\llbracket \mu v\ e \rrbracket = \llbracket \mu v\ e \rrbracket \{s/v\}$. Hence, $v$ is absent in $\llbracket \mu v\ e \rrbracket$.

- Now assume $u \neq v$. This means that $v$ is free in $\mu u\ e$ if and only if it is free in $e$, so by the induction hypothesis $v$ is absent in $\llbracket e \rrbracket$. This time, we use the induction hypothesis and write $\zeta(\llbracket e \rrbracket) = p(\vec{u}, (a_1, t_1), \ldots, (a_m, t_m))$, in which $v$ does not appear and in which $v$ is absent in each of the $s_i$.
  First consider the case where $u$ is not free in $s$. In this case, we have

$$\begin{aligned}
&\zeta(\llbracket \mu u\ e \rrbracket \{s/v\}) \\
&= \mathsf{fp}\, u\ p(\vec{u}, (a_1, s_1\{\llbracket \mu u\ e \rrbracket /u\}\{s/v\}), \ldots, (a_n, s_n\{\llbracket \mu u\ e \rrbracket /u\}\{s/v\})) \\
&\qquad\qquad \text{(def. of } \llbracket \mu u\ - \rrbracket, \{-/v\}) \\
&= \mathsf{fp}\, u\ p(\vec{u}, (a_1, s_1\{s/v\}\{\llbracket \mu u\ e \rrbracket \{s/v\}/u\}), \ldots, (a_n, s_n\{s/v\}\{\llbracket \mu u\ e \rrbracket \{s/v\}/u\})) \\
&\qquad\qquad \text{(Lemma 4.3.25)} \\
&= \mathsf{fp}\, u\ p(\vec{u}, (a_1, s_1\{\llbracket \mu u\ e \rrbracket \{s/v\}/u\}), \ldots, (a_n, s_n\{\llbracket \mu u\ e \rrbracket \{s/v\}/u\})) \\
&\qquad\qquad (v \text{ absent in } s_i) \\
&= \mathsf{fp}\, u\ \zeta(\llbracket e \rrbracket)\{\llbracket \mu u\ e \rrbracket \{s/v\}//u\})
\end{aligned}$$

  Again, $r = \llbracket \mu u\ e \rrbracket \{s/v\}$ solves the equation characterizing $\llbracket \mu u\ e \rrbracket$, so by Lemma 4.3.10, $\llbracket \mu u\ e \rrbracket = \llbracket \mu u\ e \rrbracket \{s/v\}$.
  Finally, consider the following. Both $u$ and $v$ are clearly absent in $\llbracket 0 \rrbracket$, so for arbitrary $s \in Z$ we have

$$\begin{aligned}
\llbracket \mu u\ e \rrbracket \{s/v\} = \llbracket \mu u\ e \rrbracket \{\llbracket 0 \rrbracket /v\}\{s/v\} &= \llbracket \mu u\ e \rrbracket \{\llbracket 0 \rrbracket \{s/v\}/v\} \\
= \llbracket \mu u\ e \rrbracket \{\llbracket 0 \rrbracket /v\} &= \llbracket \mu u\ e \rrbracket
\end{aligned}$$

It follows that $v$ is absent in $[\![\mu u\ e]\!]$. □

This brings us to the proof of Lemma 4.3.7, which states that for any $e, g \in Exp$ and $v \in Var$ such that $e[g/v]$ is defined, $[\![e[g/v]]\!] = [\![e]\!]\{[\![g]\!]/v\}$.

*Proof of Lemma 4.3.7.* We proceed by induction on $e$.

- For the variable case, let $u \neq v$. There are two cases to consider. First, suppose $e = v$. We have $\zeta([\![v]\!]\{[\![g]\!]/v\}) = \zeta([\![g]\!]) = \zeta([\![v[g/v]]\!])$. Now assume $e = u$. Here, we have $\zeta([\![u]\!]\{[\![g]\!]/v\}) = \zeta([\![u]\!]) = \zeta([\![u[g/v]]\!])$.
- For the inductive step, assume the result for $e$ and consider the process term $ae$. We have

$$\zeta([\![ae]\!]\{[\![g]\!]/v\}) = (a, [\![e]\!]\{[\![g]\!]/v\}) = (a, [\![e[g/v]]\!]) = \zeta([\![ae[g/v]]\!])$$

- Now consider $\sigma(e_1, \ldots, e_n)$ for $e_i \in Exp$, $i \leq n$, and assume the result for $e_1, \ldots, e_n$. We have

$$\begin{aligned}\zeta([\![\sigma(e_1,\ldots,e_n)]\!]\{[\![g]\!]/v\}) &= \sigma(\zeta([\![e_1]\!]\{[\![g]\!]/v\}), \ldots, \zeta([\![e_n]\!]\{[\![g]\!]/v\})) \\ &= \sigma(\zeta([\![e_1[g/v]]\!]), \ldots, \zeta([\![e_n[g/v]]\!])) \\ &= \zeta([\![\sigma(e_1,\ldots,e_n)[g/v]]\!])\end{aligned}$$

- Finally, assume the result for $e$ and consider $\mu u\ e$. Assume that $\zeta([\![e]\!]) = p(u, \vec{v}, (a_1, q_1), \ldots, (a_n, q_n))$. Since no free variable of $g$ is bound in $\mu u\ e$, $u$ in particular is not free in $g$. By Lemma 4.3.30, $u$ is therefore absent in $[\![g]\!]$, so

$$\begin{aligned}&\zeta([\![\mu u\ e]\!]\{[\![g]\!]/v\}) \\ &= \mathsf{fp}\, u\ p(u, \vec{v}, (a_1, t_1\{[\![\mu u\ e]\!]/u\}\{[\![g]\!]/v\}), \ldots, (a_m, t_m\{[\![\mu u\ e]\!]/u\}\{[\![g]\!]/v\})) \\ &= \mathsf{fp}\, u\ p(u, \vec{v}, (a_1, t_1\{[\![g]\!]/v\}\{[\![\mu u\ e]\!]\{[\![g]\!]/v\}/u\}), \\ &\qquad\qquad \ldots, (a_m, t_m\{[\![g]\!]/v\}\{[\![\mu u\ e]\!]\{[\![g]\!]/v\}/u\})) \quad \text{(Lemma 4.3.25)} \\ &= \mathsf{fp}\, u\ \zeta([\![e]\!]\{[\![g]\!]/v\})\{[\![\mu u\ e]\!]\{[\![g]\!]/v\}//u\} \\ &= \mathsf{fp}\, u\ \zeta([\![e[g/v]]\!])\{[\![\mu u\ e]\!]\{[\![g]\!]/v\}//u\} \quad \text{(ind. hyp.)}\end{aligned}$$

Hence, $[\![\mu u\ e]\!]\{[\![g]\!]/v\}$ satisfies the equation characterizing $[\![\mu u\ (e[g/v])]\!]$. It

follows from Lemma 4.3.10 that

$$[\![\mu u\ (e[g/v])]\!] = [\![(\mu u\ e)[g/v]]\!] = [\![\mu u\ e]\!]\{[\![g]\!]/v\} \qquad \square$$

## 4.4 An Axiomatization of Behavioural Equivalence

An important corollary of Theorem 4.3.9 is that behavioural equivalence is a $\Sigma_M$-congruence on $(Exp, \mathsf{ev})$, meaning that it is preserved by all the program constructs of $\Sigma_M$. This opens the door to the possibility of algebraically reasoning about behavioural equivalences between process terms from just a few axioms. The purpose of this section is to show that all behavioural equivalences between process terms can be derived from the equations EQ presenting $(M, \eta, \mu)$ as well as three axiom schemas concerning the recursion operators. We are specifically going to focus on process algebras obtained from iterative branching theories, as they are the most common type of process calculus that appears in the literature (see Examples 4.1.7 and 4.2.7 to 4.2.9).

**Definition 4.4.1.** Given an iterative branching theory $(S, \mathsf{EQ}, \mathsf{fp})$, the *effectful process calculus* obtained from $(S, \mathsf{EQ}, \mathsf{fp})$ is the $\Sigma_M$-algebra $(Exp/\equiv, [\mathsf{ev}]_\equiv)$ of process terms modulo the theory EFA, which consists of the axioms of EQ and (R1)-(R4) below.

The first two out of the three recursion axiom schemas for the resulting specification language are

$$\text{(R1)} \quad \mu v\ e = e[\mu v\ e/v] \qquad \text{(R2)} \quad \frac{w \text{ not free in } e}{\mu v\ e = \mu w\ (e[w/v])}$$

The axiom (R1) allows us to unravel recursive terms. This has the effect of identifying $\mu v\ av$ with $a(\mu v\ av)$, for example. This satisfies our intuition that $\mu v\ av$ should solve the recursive specification $x = ax$ in the indeterminate $x$.

The axiom (R2) states that equivalence is invariant under $\alpha$-conversion. That is, recursion variables can be swapped for fresh variables without changing the congruence class. This identifies process terms like $\mu v\ av$ and $\mu w\ aw$, which intuitively should denote the same solution to $x = ax$.

The third and fourth recursion axiom schemas can be stated as proof rules,

$$\text{(R3)}\quad \frac{v \text{ guarded in } g_1,\ldots,g_n}{\mu v\ p(v,\vec{g}) = \mu v\ (\mathsf{fp}\ v\ p(v,\vec{g}))} \qquad \text{(R4)}\quad \frac{g = e[g/v] \quad v \text{ guarded in } e}{g = \mu v\ e}$$

The first axiom tell us that $\mu v$ computes fixed-points by applying the unguarded iteration operator first, and the second axiom that the fixed-point denoted by $\mu v$ is unique for guarded expressions.

**Definition 4.4.2.** For $e, f \in Exp$, if $e = f$ is derivable from EFA and equational logic (extended to the algebraic signature $\Sigma_M$), then we write $\mathsf{EFA} \vdash e = f$ or $e \equiv f$ and say that $e$ and $f$ are *provably equivalent*. We write $[-]_\equiv : Exp \to Exp/\equiv$ for the quotient map $Exp \to Exp/\equiv$.

When we refer to examples like ARB, ACF, APA, and ARB, we are often identifying each of these with their associated effectful process algebra $(Exp/\equiv, [\mathsf{ev}]_\equiv)$ of process terms modulo $\equiv$.

**Soundness**

Fix an iterative branching theory $(S, \mathsf{EQ}, \mathsf{fp})$ presenting a monad $(M, \eta, \mu)$. We would like to argue that $\equiv$ is *sound* with respect to behavioural equivalence of $B_M$-coalgebras, that $[\![e]\!] = [\![f]\!]$ whenever $e \equiv f$. This can be derived from the fact that the set of congruence classes of process terms itself forms a $B_M$-coalgebra.

**Lemma 4.4.3.** *The congruence $\equiv$ is the kernel of a coalgebra homomorphism.*

Throughout the following proof, we allow for the lifting of $[-//v]$ to $Exp/\equiv$, defined inductively and satisfying $B_M([-]_\equiv)(p)[[g]_\equiv//v] = B_M([-]_\equiv)(p[g//v])$.

*Proof.* We need to fill in the following diagram with a map $\bar{\varepsilon}$

$$\begin{array}{ccc} Exp & \xrightarrow{[-]_\equiv} & Exp/\equiv \\ {\scriptstyle \mathsf{ev}}\downarrow & & \vdots\,{\scriptstyle \bar{\varepsilon}} \\ B_M Exp & \xrightarrow{B_M([-]_\equiv)} & B_M(Exp/\equiv) \end{array} \tag{4.10}$$

Since $[-]_\equiv$ is surjective, we would like to be able to define $\bar{\varepsilon}$ by setting $\bar{\varepsilon}([e]_\equiv) = B_M([-]_\equiv) \circ \varepsilon(e)$ for any $e \in Exp$. That this is well-defined amounts[10] to showing the

[10] This is called the *diagonal fill-in property* of **Set** [Rut00].

inclusion $\ker([-]_{\equiv}) \subseteq \ker(B_M([-]_{\equiv}) \circ \varepsilon)$. Since $\ker([-]_{\equiv})$ is $\equiv$, we need to show that $e \equiv f$ implies $B_M([-]_{\equiv}) \circ \varepsilon(e) = B_M([-]_{\equiv}) \circ \varepsilon(f)$. We do this by induction on the proof of $\mathsf{EFA} \vdash e = f$.

If $(e,f) \in \mathsf{EQ}$, then since $\varepsilon$ is an $S$-algebra homomorphism, $\varepsilon(e) = \varepsilon(f)$. It is trivial that (Ref) is sound. The other rules that do not have any premises are (R1) and (R2). In (R1), let $\varepsilon(e) = p(v, \vec{u}, \vec{\xi})$ where $\xi_i \in Act \times Exp$.

$$
\begin{aligned}
B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ e) &= B_M([-]_{\equiv})\big(\mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]\big) \\
&= B_M([-]_{\equiv})\big(\mathsf{fp}\ v\ \varepsilon(e)\big)[[\mu v\ e]_{\equiv}//v] \\
&= B_M([-]_{\equiv})\big(p(\mathsf{fp}\ v\ \varepsilon(e), \vec{u}, \vec{\xi})\big)[[\mu v\ e]_{\equiv}//v] && (\mathsf{fp}\text{ is iter.}) \\
&= \big(p(B_M([-]_{\equiv})(\mathsf{fp}\ v\ \varepsilon(e)), \vec{u}, B_M([-]_{\equiv})(\vec{\xi}))\big)[[\mu v\ e]_{\equiv}//v] \\
& && (\text{homom.}) \\
&= p(B_M([-]_{\equiv})(\mathsf{fp}\ v\ \varepsilon(e))[[\mu v\ e]_{\equiv}//v], \vec{u}, B_M([-]_{\equiv})(\vec{\xi})[[\mu v\ e]_{\equiv}//v]) \\
& && (\text{homom.}) \\
&= p(B_M([-]_{\equiv})(\mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]), \vec{u}, B_M([-]_{\equiv})(\vec{\xi}[\mu v\ e//v])) \\
&= p(B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ e), \vec{u}, B_M([-]_{\equiv})(\vec{\xi}[\mu v\ e//v])) && (\text{def. } \mu v) \\
&= B_M([-]_{\equiv})\big(p(\varepsilon(\mu v\ e), \vec{u}, \vec{\xi}[\mu v\ e//v])\big) \\
&= B_M([-]_{\equiv}) \circ \varepsilon(e[\mu v\ e/v])
\end{aligned}
$$

Soundness of the rule (R2) is a consequence of the fact that $\mathsf{fp}\ v\ p(v,\vec{x}) = \mathsf{fp}\ w\ p(w,\vec{x})$ if $w \notin \vec{x}$, and that $f[g/v] = f[w/v][g/w]$. This concludes the base case.

For the inductive case, assume that $e \equiv f$, $e_i \equiv f_i$, and let $a \in Act$, $v \in Var$. Again, (Ref) and (Tra) trivially hold. Next we show that (Con) holds: In the $Act \times Exp$ case,

$$B_M([-]_{\equiv}) \circ \varepsilon(ae) = (a, [e]_{\equiv}) = (a, [f]_{\equiv}) = B_M([-]_{\equiv}) \circ \varepsilon(af)$$

In the $S$-algebra case,

$$
\begin{aligned}
B_M([-]_{\equiv}) \circ \varepsilon(\sigma(\vec{e})) &= \sigma(B_M([-]_{\equiv}) \circ \varepsilon(e_1), \ldots, B_M([-]_{\equiv}) \circ \varepsilon(e_n)) \\
&= \sigma(B_M([-]_{\equiv}) \circ \varepsilon(f_1), \ldots, B_M([-]_{\equiv}) \circ \varepsilon(f_n)) && (\text{ind. hyp.}) \\
&= B_M([-]_{\equiv}) \circ \varepsilon(\sigma(\vec{f}))
\end{aligned}
$$

In the recursion case,

$$\begin{aligned}
B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ e) &= B_M([-]_{\equiv})(\mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]) \\
&= \mathsf{fp}\ v\ B_M([-]_{\equiv})(\varepsilon(e))[[\mu v\ e]_{\equiv}//v] \\
&= \mathsf{fp}\ v\ B_M([-]_{\equiv})(\varepsilon(f))[[\mu v\ e]_{\equiv}//v] && \text{(induct. hyp.)} \\
&= \mathsf{fp}\ v\ B_M([-]_{\equiv})(\varepsilon(f))[[\mu v\ f]_{\equiv}//v] && \text{(Con)} \\
&= B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ f)
\end{aligned}$$

For (R3), observe that if $v$ is guarded in $g_1, \ldots, g_n$, then

$$\begin{aligned}
\varepsilon(\mu v\ (p(v,\vec{g}))) &= \mathsf{fp}\ v\ \varepsilon(p(v,\vec{g}))[\mu v\ p(v,\vec{g})//v] && \text{(def.)} \\
&= \mathsf{fp}\ v\ p(v,\varepsilon(\vec{g}))[\mu v\ p(v,\vec{g})//v] && (\varepsilon \text{ an } S\text{-alg. homom.}) \\
&= \varepsilon(\mathsf{fp}\ v\ p(v,\vec{g}))[\mu v\ p(v,\vec{g})//v]
\end{aligned}$$

so that under $B_M([-]_{\equiv})$,

$$\begin{aligned}
B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ (p(v,\vec{g}))) &= B_M([-]_{\equiv})(\varepsilon(\mathsf{fp}\ v\ p(v,\vec{g}))[\mu v\ p(v,\vec{g})//v]) \\
&= B_M([-]_{\equiv}) \circ \varepsilon(\mathsf{fp}\ v\ p(v,\vec{g}))[[\mu v\ p(v,\vec{g})]_{\equiv}//v] \\
&= B_M([-]_{\equiv}) \circ \varepsilon(\mathsf{fp}\ v\ p(v,\vec{g}))[[\mu v\ (\mathsf{fp}\ v\ p(v,\vec{g}))]_{\equiv}//v] && \text{(R3)} \\
&= B_M([-]_{\equiv})(\varepsilon(\mathsf{fp}\ v\ p(v,\vec{g}))[\mu v\ (\mathsf{fp}\ v\ p(v,\vec{g}))//v]) \\
&= B_M([-]_{\equiv})(\varepsilon(\mu v\ (\mathsf{fp}\ v\ p(v,\vec{g}))))
\end{aligned}$$

The last equality holds because if $v \in \mathrm{gv}(e)$, then $\varepsilon(\mu v\ e) = \varepsilon(e)[\mu v\ e/v]$, for any $e \in \mathit{Exp}$. Finally, for (R4), assume that $v$ is guarded in $e$, $g \equiv e[g//v]$, and that

$$B_M([-]_{\equiv}) \circ \varepsilon(g) = B_M([-]_{\equiv}) \circ \varepsilon(e[g/v])$$

Then we can derive

$$\begin{aligned}
B_M([-]_{\equiv}) \circ \varepsilon(e[g/v]) &= B_M([-]_{\equiv})(\varepsilon(e)[g//v]) && (v \in \mathrm{gv}(e)) \\
&= B_M([-]_{\equiv})(\varepsilon(e))[[g]_{\equiv}//v] \\
&= B_M([-]_{\equiv})(\varepsilon(e))[[\mu v\ e]_{\equiv}//v] && \text{(R4)}
\end{aligned}$$

$$
\begin{aligned}
&= B_M([-]_{\equiv})(\varepsilon(e)[\mu v\ e//v]) \\
&= B_M([-]_{\equiv})(\mathsf{fp}\ v\ \varepsilon(e)[\mu v\ e//v]) \qquad (*) \\
&= B_M([-]_{\equiv}) \circ \varepsilon(\mu v\ e)
\end{aligned}
$$

In (*), we used that $v$ does not appear as a free variable in an $S$-term representing $\varepsilon(e)$, due to $v$ being guarded in $e$ (this can be shown easily by induction on $e$). □

We write $(Exp/{\equiv}, \bar{\varepsilon})$ for the coalgebra structure on $Exp/{\equiv}$ making $[-]_{\equiv}$ a coalgebra homomorphism (there is one such coalgebra structure [Rut00]). As $[\![-]\!]$ is the unique coalgebra homomorphism $(Exp, \varepsilon) \to (Z, \zeta)$, and because there is also a coalgebra homomorphism $!_{\bar{\varepsilon}} : (Exp/{\equiv}, \bar{\varepsilon}) \to (Z, \zeta)$, it must be the case that $!_{\bar{\varepsilon}} \circ [-]_{\equiv} = [\![-]\!]$. By Lemma 4.4.3, if $e \equiv f$, then $[\![e]\!] = !_{\bar{\varepsilon}}([e]_{\equiv}) = !_{\bar{\varepsilon}}([f]_{\equiv}) = [\![f]\!]$. This establishes the following.

**Theorem 4.4.4** (Soundness)**.** *Let $e, f \in Exp$. If $e \equiv f$, then $[\![e]\!] = [\![f]\!]$.*

Soundness allows us to derive at least a subset of all the behavioural equivalences between process terms from the axioms in EQ and R. If our aspiration were simply to have a set of behaviour-preserving code-transformations, then we could simply stop here and be satisfied, since in principle we could see the axioms of EQ and (R1)-(R4) as rewrite rules that satisfy this purpose.

### Completeness

Aiming a bit higher than deriving only a few behavioural equivalences between process terms, we move on to show the converse of Theorem 4.4.4, that $\equiv$ is *complete* with respect to behavioural equivalence. We make use of Lemma 2.5.3 below, which allows us to organize the completeness proof into a few intuitive steps.

**Lemma 2.5.3.** *Let $(E, \varepsilon)$ and $(Z, \zeta)$ be $B$-coalgebras with $(Z, \zeta)$ final. Then the behaviour map $!_{\varepsilon} : (E, \varepsilon) \to (Z, \zeta)$ is injective if and only if there is a class $\mathbf{C}$ of $B$-coalgebras that is closed under homomorphic images and such that $(E, \varepsilon)$ is final in $\mathbf{C}$.*

A *subcoalgebra* of a $B$-coalgebra $(X, \beta)$ is an injective map $\mathrm{in} : U \hookrightarrow X$ such that $\beta|_U$ factors through $B(\mathrm{in})$. A $B$-coalgebra is *locally finite* if every of its states is contained in (the image of) a finite subcoalgebra. We instantiate Lemma 2.5.3 in the case where $B = B_M$, $(E, \varepsilon) = (Exp/{\equiv}, \bar{\varepsilon})$, and $\mathbf{C}$ is the class of locally finite $B_M$-coalgebras.

Completeness of $\equiv$ with respect to behavioural equivalence follows shortly after, for if $[\![e]\!] = [\![f]\!]$, then $!_{\bar{\varepsilon}}([e]_\equiv) = !_{\bar{\varepsilon}}([f]_\equiv)$. By Lemma 2.5.3, $!_{\bar{\varepsilon}}$ is injective, so $[e]_\equiv = [f]_\equiv$ or equivalently, $e \equiv f$. To establish the converse of Theorem 4.4.4, it suffices to show that our choices of $(E, \varepsilon)$ and $\mathbf{C}$ satisfy the hypotheses of Lemma 2.5.3.

**Lemma 4.4.5.** *The coalgebra $(Exp, \varepsilon)$ is locally finite.*

*Proof.* Given $e \in Exp$, we construct a subcoalgebra of $(Exp, \varepsilon)$ that has a finite set of states that includes $e$. To this end, define $U : Exp \to \mathcal{P}_\omega(Exp)$ by

$$U(v) = \{v\} \qquad U(ae) = \{ae\} \cup U(e) \qquad U(\sigma(e_1, \ldots, e_n)) = \{\sigma(e_1, \ldots, e_n)\} \cup \bigcup_{i \le n} U(e_i)$$

$$U(\mu v\ e) = \{\mu v\ e\} \cup U(e)[\mu v\ e//v] = \{\mu v\ e\} \cup \{f[\mu v\ e//v] \mid f \in U(e)\}$$

Note that $e \in U(e)$ for all $e \in Exp$ and that $U(e)$ is finite. We begin with following claim, which says that the outgoing transitions of $e$ are given in terms of expressions from $U(e)$: For any $e \in Exp$, there is a representative $S$-term $p \in \varepsilon(e)$ such that $p \in S^*(Var + Act \times U(e))$. This can be seen by induction on the construction of $e$, the only interesting case being in the inductive step $\mu v\ e$. Here, let $\varepsilon(e) = q$ and observe that $p = \mathsf{fp}\, v\ q[\mu v\ e//v]$ is a representative of $\varepsilon(\mu v\ e)$ in $S^*(Var + Act \times U(\mu v\ e))$.

To finish the proof of the lemma, fix an $e \in Exp$ and define a sequence of sets beginning with $U_0 = \{e\}$ and proceeding with

$$U_{n+1} = U_n \cup \bigcup_{e_0 \in U_n} \{g \mid (\exists a \in Act)(\exists p \in \varepsilon(e_0) \cap S^*(Var + Act \times U(e_0)))\ (a, g) \text{ appears in } p\}$$

Then $U_0 \subseteq U_1 \subseteq \cdots \subseteq U(e)$ and the latter set is finite, so $U := \bigcup U_n$ is finite and contained in $U(e)$. We define a coalgebra structure $\varepsilon_U : U \to B_M U$ by taking $\varepsilon_U(e) = [p]_{\mathsf{EQ}}$ where if $e \in U_n$, then $p$ is a representative of $\varepsilon(e)$ in $S^*(Var + Act \times U_{n+1})$. Since $S^*(Var + Act \times U_{n+1}) \subseteq S^*(Var + Act \times U)$, this defines a $B_M$-coalgebra structure on $U$. Where $\mathrm{in} : U \hookrightarrow Exp$, we have $\varepsilon(\mathrm{in}(e)) = \varepsilon(e) = B_M(\mathrm{in}) \circ \varepsilon_U(e)$, so $(U, \varepsilon_U)$ is a finite subcoalgebra of $(Exp, \varepsilon)$ containing $e$. □

The class of locally finite coalgebras is closed under homomorphic images: If $(X, \beta)$ is locally finite and $h : (X, \beta) \to (Y, \vartheta)$ is a surjective homomorphism, then for any $y \in Y$ and $x \in X$ such that $h(x) = y$, and for any finite subcoalgebra $U$ of $(X, \beta)$ containing $x$, $h[U]$ is a finite subcoalgebra of $(Y, \vartheta)$ containing $y$ (see Lemma 2.2.9

or [Gum99]). Since $y$ was arbitrary, $(Y,\theta)$ is locally finite. It follows from Lemma 4.4.3 that $(Exp/{\equiv},\bar{\varepsilon})$ is locally finite.

What remains to be seen among the hypotheses of Lemma 2.5.3 is that $(Exp/{\equiv},\bar{\varepsilon})$ is the *final* locally finite coalgebra, meaning that for any locally finite coalgebra $(X,\beta)$ there is a unique coalgebra homomorphism $(X,\beta)\to(Exp/{\equiv},\bar{\varepsilon})$. In fact, it suffices to see that every finite subcoalgebra of $(X,\beta)$ admits a unique coalgebra homomorphism into $(Exp/{\equiv},\bar{\varepsilon})$. To see why it suffices, define $\phi_U\colon (U,\beta_U)\to(Exp/{\equiv},\bar{\varepsilon})$ to be the unique homomorphism from any finite subcoalgebra $U$ of $(X,\beta)$. Given two finite subcoalgebras $U,V$ of $(X,\beta)$ containing a particular state $x\in X$, one can find a subcoalgebra $W\subseteq U\cap V$ containing $x$ [Gum99]. By uniqueness, we have $\phi_U(x)=\phi_W(x)=\phi_V(x)$, so setting $\phi(x)=\phi_U(x)$, for any $U$ a subcoalgebra of $(X,\beta)$ containing $x$, defines a coalgebra homomorphism $\phi\colon (X,\beta)\to(Exp/{\equiv},\bar{\varepsilon})$. Furthermore, given any other coalgebra homomorphism $\psi$ of the same type, restricting to a subcoalgebra $U$ containing $x$ gives $\psi(x)=\psi_U(x)=\phi_U(x)=\phi(x)$. Hence, from uniqueness of homomorphisms from finite subcoalgebras we obtain uniqueness of homomorphisms from locally finite coalgebras.

To this end, we make use of an old idea originating in the work of Salomaa [Sal66]. We associate with every finite coalgebra a certain system of equations whose solutions (in $Exp/{\equiv}$) are in one-to-one correspondence with coalgebra homomorphisms into $(Exp/{\equiv},\bar{\varepsilon})$. Essentially, if a system admits a unique solution, then its corresponding coalgebra admits a unique homomorphism into $(Exp/{\equiv},\bar{\varepsilon})$. This would then establish finality.

**Definition 4.4.6.** A *(finite) system of equations* is a sequence of the form $\{x_i=e_i\}_{i\leq n}$ where $x_i\in Var$ and $e_i\in Exp$ for $i\leq n$, and none of $x_1,\ldots,x_n$ appear as bound variables in any of $e_1,\ldots,e_n$. A system of equations $\{x_i=e_i\}_{i\leq n}$ is *guarded* if $x_1,\ldots,x_n$ are guarded in every $e_i$. A *solution* to $\{x_i=e_i\}_{i\leq n}$ is a function $\varphi\colon\{x_i\}_{i\leq n}\to Exp$ such that

$$\varphi(x_i)\equiv e_i[\varphi(x_1)/x_1,\ldots,\varphi(x_n)/x_n]$$

for all $i\leq n$ and $x_1,\ldots,x_n$ do not appear free in $\varphi(x_i)$ for any $i\leq n$.

Every finite $B_M$-coalgebra $(X,\beta)$ gives rise to a guarded system of equations in

the following way: for each $p \in S^*(Var + Act \times X)$, define $p^\dagger$ inductively as

$$v^\dagger = v \qquad (a,e)^\dagger = ae \qquad \sigma(q_1,\ldots,q_n)^\dagger = \sigma(q_1^\dagger,\ldots,q_n^\dagger)$$

and for each $x \in X$, let $p_x$ be a representative of $\beta(x)$. The[11] system of equations *associated with* $(X,\beta)$ is then defined to be $\{x = p_x^\dagger\}_{x\in X}$. We treat the elements of $X$ as variables in these equations. By definition, every $y \in X$ is guarded in $p_x^\dagger$.

**Theorem 4.4.7.** *Let $(X,\beta)$ be a finite $B_M$-coalgebra and $\varphi\colon X \to Exp$ a function. Then the composition $[-]_\equiv \circ \varphi\colon X \to Exp/\equiv$ is a $B_M$-coalgebra homomorphism if and only if $\varphi$ is a solution to the system of equations associated with $(X,\beta)$.*

The proof of this theorem requires a lemma in the spirit of the fundamental theorem of regular expressions (Theorem 2.2.18).

**Lemma 4.4.8.** *For any $e \in Exp$, there exist $v_1,\ldots,v_n \in Var$, and process terms $g_1,\ldots,g_m$ such that $v_i$ is guarded in $g_j$ for all $i,j$, and there exists some $p(\vec{x},\vec{y}) \in S^*\{x_1,\ldots,x_n,y_1,\ldots,y_m\}$ such that $\mathsf{EFA} \vdash e = p(\vec{v},\vec{g})$.*

*Proof.* The proof is by induction on $e$, the only interesting case being the inductive step for the recursion operator $\mu v\ e$. First, find $q(v,\vec{v},\vec{f}) \equiv e$ using the induction hypothesis, then use (R3). Since $v$ is not free in $\mathsf{fp}\, v\ q(v,\vec{v},\vec{f})$, we have

$$\begin{aligned}
\mathsf{EFA} \vdash \mu v\ e &= \mu v\ q(v,\vec{v},\vec{f}) \\
&= \mu v\ (\mathsf{fp}\, v\ q(v,\vec{v},\vec{f})) && \text{(R3)} \\
&= \mathsf{fp}\, v\ q(v,\vec{v},\vec{f}[\mu v\ (\mathsf{fp}\, v\ q(v,\vec{v},\vec{f}))/v]) && \text{(R1)}
\end{aligned}$$

Take $p(\vec{x},\vec{y}) = \mathsf{fp}\, v\ q(v,\vec{x},\vec{y})$, $\vec{g} = \vec{f}[\mu v\ (\mathsf{fp}\, v\ q(v,\vec{v},\vec{f}))/v]$. □

The consequence of Lemma 4.4.8 that is used in the proof of Theorem 4.4.7 is that for any $e \in Exp$ and $v \in Var$, there is an $e' \in Exp$ such that $v$ is guarded in $e'$ and $\mathsf{EFA} \vdash \mu v\ e = \mu v\ e'$.

*Proof of Theorem 4.4.7.* We begin by observing that $\bar{\varepsilon}\colon Exp/\equiv \to B_M Exp/\equiv$ is a bijec-

[11] Technically speaking, there could be many systems of equations associated with a given coalgebra. We say "the" system of equations because any two have the same set of solutions up to EQ.

tion. Indeed, the map $(-)^{\heartsuit}\colon B_M Exp/{\equiv} \to Exp/{\equiv}$ defined

$$v^{\heartsuit} = [v]_{\equiv} \qquad (a,[e]_{\equiv})^{\heartsuit} = [ae]_{\equiv} \qquad \sigma(p_1,\dots,p_n)^{\heartsuit} = \sigma(p_1^{\heartsuit},\dots,p_n^{\heartsuit})$$

is its inverse. By construction, $\bar{\varepsilon}(p^{\heartsuit}) = p$ for any $p \in B_M Exp/{\equiv}$, so it suffices to see that $\bar{\varepsilon}([e]_{\equiv})^{\heartsuit} = [e]_{\equiv}$ for all $e \in Exp$. This can be done by induction on the construction of $e$, and again the only interesting case is $\mu v\ e$. By Lemma 4.4.8, we can assume that $v$ is guarded in $e$. Furthermore, we can define $[g]_{\equiv}[[\mu v\ e]_{\equiv}//v] = [g[\mu v e//v]]$ on $Exp/{\equiv}$ by (Con).

$$\begin{aligned}
\bar{\varepsilon}([\mu v\ e]_{\equiv})^{\heartsuit} &= (\bar{\varepsilon}(e)[[\mu v\ e]_{\equiv}//v])^{\heartsuit} \\
&= (\bar{\varepsilon}(e))^{\heartsuit}[[\mu v\ e]_{\equiv}//v] && (*)\\
&= [e]_{\equiv}[[\mu v\ e]_{\equiv}//v] && \text{(induct. hyp.)}\\
&= [e[\mu v\ e/v]]_{\equiv} \\
&= [\mu v\ e]_{\equiv} && \text{(R1)}
\end{aligned}$$

I used that guarded syntactic substitution commutes with $(-)^{\heartsuit}$ in (*), which can be seen with an easy induction on terms in $S^*(Var + Act \times (Exp/{\equiv}))$.

Now let $\{x = p_x^{\dagger}\}_{x\in X}$ be the system of equations associated with the coalgebra $(X,\beta)$. Observe that for any $x,y \in X$, if $y$ appears in $p_x$, then it is guarded in $p_x^{\dagger}$. This means that $\varphi\colon X \to Exp$ is a solution to $\{x = p_x^{\dagger}\}_{x\in X}$ if and only if $\varphi(x) \equiv p_x^{\dagger}[\varphi(y)//y]_{y\in X}$. Now, if $\beta(x) = p_x$, we see that

$$\begin{aligned}
(B_M([-]_{\equiv}\circ\varphi)(\beta(x)))^{\heartsuit} &= (B_M([-]_{\equiv})\circ B_M(\varphi)(p_x))^{\heartsuit} && \text{(functoriality)}\\
&= (B_M([-]_{\equiv})(p_x[\varphi(y)//y]_{y\in X}))^{\heartsuit} && (B_M(\varphi) \text{ an alg. homom.})\\
&= (p_x[\varphi(y)//y]_{y\in X})^{\heartsuit} \\
&= p_x^{\dagger}[\varphi(y)//y]_{y\in X}
\end{aligned}$$

Thus, $\varphi$ is a solution to the system $\{x = p_x^{\dagger}\}_{x\in X}$ if and only if

$$[-]_{\equiv}\circ\varphi(x) = (-)^{\heartsuit}\circ B_M([-]_{\equiv}\circ\varphi)\circ\beta(x) \tag{4.11}$$

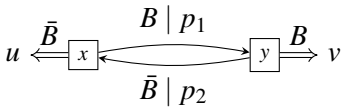
for every $x \in X$. The maps $(-)^{\heartsuit}$ and $\bar{\varepsilon}$ are inverse to one another, so (4.11) is

equivalent to the identity $\bar{\varepsilon} \circ [-]_{\equiv} \circ \varphi = B_M([-]_{\equiv} \circ \varphi) \circ \beta$. This identity is the defining property of a coalgebra homomorphism of the form $[-]_{\equiv} \circ \varphi$. □

As a direct consequence of Theorem 4.4.7, we see that a finite subcoalgebra $U \hookrightarrow Exp$ of $(Exp, \varepsilon)$ is a solution to the system of equations associated with $(U, \varepsilon|_U)$.

*Example* 4.4.9. The system of equations associated with the automaton in Example 4.2.7 is the two-element set consisting of

$$x = p_1 y +_B u \qquad y = v +_B p_2 x$$




The map $\varphi\colon \{x_1, x_2\} \to Exp$ defined by $\varphi(x_1) = \mu w\ (p_1(v +_B p_2 w) +_B u)$ and $\varphi(x_2) = v +_B p_2\ \varphi(x_1)$ is a solution.

Theorem 4.4.7 establishes a one-to-one correspondence between solutions to systems and coalgebra homomorphisms as follows. Say that two solutions $\varphi$ and $\psi$ to a system $\{x_i = e_i\}_{i \leq n}$ are $\equiv$-*equivalent* if $\varphi(x_i) \equiv \psi(x_i)$ for all $i \leq n$. Starting with a solution $\varphi\colon X \to Exp$ to the system associated with $(X, \beta)$, we obtain the homomorphism $[-]_{\equiv} \circ \varphi$ using Theorem 4.4.7. A pair of solutions $\varphi$ and $\psi$ are $\equiv$-equivalent if and only if $[-]_{\equiv} \circ \varphi = [-]_{\equiv} \circ \psi$, so up to $\equiv$-equivalence the correspondence $\varphi \mapsto [-]_{\equiv} \circ \varphi$ is injective. Going in the opposite direction and starting with a homomorphism $\psi\colon (X, \beta) \to (Exp/{\equiv}, \bar{\varepsilon})$, let $e_x$ be a representative of $\psi(x)$ for each $x \in X$ and define $\varphi = \lambda x.e_x$. Then $\varphi$ is a solution to $(X, \beta)$, and $[-]_{\equiv} \circ \varphi = \psi$. Thus, up to $\equiv$-equivalence, solutions to systems are in one-to-one correspondence with coalgebra homomorphisms into $(Exp/{\equiv}, \bar{\varepsilon})$.

Say that a system *admits a unique solution up to* $\equiv$ if it has a solution and any two solutions to the system are $\equiv$-equivalent. Since, up to $\equiv$-equivalence, solutions to a system associated with a coalgebra $(X, \beta)$ are in one-to-one correspondence with coalgebra homomorphisms $(X, \beta) \to (Exp/{\equiv}, \bar{\varepsilon})$, it suffices for the purposes of satisfying the hypotheses of Lemma 2.5.3 to show that every finite guarded system of equations admits a unique solution up to $\equiv$. The following theorem is a generalization of [Mil84, Theorem 5.7].

**Theorem 4.4.10.** *Finite guarded systems of equations admit a unique solution up to* $\equiv$.

The proof is a recreation of the one that appears under [Mil84, Theorem 5.7]

with the more general context of this chapter in mind. Remarkably, the essential details of the proof remain unchanged despite the jump in abstraction.

*Proof.* Let $\{x_i = e_i\}_{i \leq n}$ be a guarded system of equations. We proceed by induction on $n$. In the base case, $n = 1$. This case is straight-forward because $\varphi(x_1) := \mu x_1\ e_1$ is its unique solution up to $\equiv$ by (R4).

Now assume that every system of strictly fewer than $n$ guarded equations has a unique solution up to $\equiv$. Define $f_n = \mu x_n\ e_n$ and $f_i = e_i[f_n/x_n]$ for each $i < n$. Since $x_1, \ldots, x_n$ are guarded in $e_i$ for $i \leq n$, the system $\{x_i = f_i\}_{i<n}$ is also guarded, and $x_1, \ldots, x_{n-1}$ do not appear freely in $f_i$ for any $i < n$. By the induction hypothesis, $\{x_i = f_i\}_{i<n}$ has a unique solution $\psi\colon \{x_1, \ldots, x_{n-1}\} \to \mathit{Exp}$ up to $\equiv$.

Let $g_i = \psi(x_i)$ for $i < n$ and $g_n = f_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}]$, and note that $x_n$ is not free and does not appear bound in any of $f_1, \ldots, f_{n-1}$ by construction. Then $\varphi(x_i) := g_i$ for $i \leq n$ is indeed a solution of the desired form, since

$$\begin{aligned}
g_n &= f_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}] \\
&= (\mu x_n\ e_n)[g_1/x_1, \ldots, g_{n-1}/x_{n-1}] \\
&= \mu x_n\ (e_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}]) && (x_n \text{ not free in } g_i) \\
&\equiv e_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}][g_n/x_n] && (\text{R1}), (x_n \text{ guarded in } g_n) \\
&= e_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}, g_n/x_n] && (x_n \text{ not free in } g_i)
\end{aligned}$$

and for any $i < n$,

$$\begin{aligned}
g_i &\equiv f_i[g_1/x_1, \ldots, g_{n-1}/x_{n-1}] \\
&= e_i[f_n/x_n][g_1/x_1, \ldots, g_{n-1}/x_{n-1}] \\
&= e_i[g_1/x_1, \ldots, g_{n-1}/x_{n-1}][f_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}]/x_n] \\
&= e_i[g_1/x_1, \ldots, g_{n-1}/x_{n-1}, f_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}]/x_n] \\
&= e_i[g_1/x_1, \ldots, g_{n-1}/x_{n-1}, g_n/x_n]
\end{aligned}$$

because $x_n$ is not free in $g_i$ for any $i < n$.

To see that $\varphi$ is the unique solution, let $\theta(x_i) = h_i$ for $i \leq n$ be any other solution

to $\{x_i = e_i\}_{i \leq n}$. Then in particular,

$$h_n \equiv e_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}, h_n/x_n] = e_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}][h_n/x_n]$$

because $x_n$ is not free in any $h_1, \ldots, h_{n-1}$. This means that

$$h_n \equiv \mu x_n \ (e_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}])$$

by (R4) and guardedness of $x_n$ in $e_n$. Since $x_n$ is not free in $h_i$ for any $i < n$,

$$\begin{aligned} \mu x_n \ (e_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}]) &= (\mu x_n \ e_n)[h_1/x_1, \ldots, h_{n-1}/x_{n-1}] \\ &= f_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}] \end{aligned}$$

This makes the restriction of $\theta$ to $x_1, \ldots, x_{n-1}$ a solution to $\{x_i = f_i\}_{i<n}$. By the induction hypothesis, there is only one such solution, so $h_i \equiv g_i$ for each $i < n$. It follows that

$$h_n \equiv \mu x_n \ (e_n[h_1/x_1, \ldots, h_{n-1}/x_{n-1}]) \equiv \mu x_n \ (e_n[g_1/x_1, \ldots, g_{n-1}/x_{n-1}]) = g_n$$

via the congruence laws. Therefore, $h_n \equiv g_n$, and overall $\theta \equiv \varphi$. □

Completeness of $\equiv$ with respect to behavioural equivalence is now a direct consequence of Lemma 2.5.3 and Theorems 4.4.7 and 4.4.10.

**Theorem 4.4.11** (Completeness)**.** *Let $e, f \in Exp$. If $[\![e]\!] = [\![f]\!]$, then $e \equiv f$.*

One way to interpret this theorem is that the algebra $(Exp/\equiv, [\mathsf{ev}]_\equiv)$ of process terms modulo $\equiv$ is isomorphic to a subalgebra of $(Z, \gamma)$, or dually $(Exp/\equiv, \bar{\varepsilon})$ is a subcoalgebra of $(Z, \zeta)$. It is in this sense that ARB, ACF, APA, and ARB are algebras of behaviours.

## 4.5 Star Fragments

In this section we study a fragment of our specification languages consisting of *star expressions*. These include primitive actions from $Act$, a form of sequential composition, and analogues of the Kleene star. We do not aim to give a complete axiomatization of behavioural equivalence for star expressions in the same style as Salomaa's axioms

in Figure 2.1, as even in simple cases this is notoriously difficult (recall the difficulty of Milner's completeness problem from Chapter 2, or the completeness problem of GKAT in Chapter 3). Nevertheless, we think it is valuable to extrapolate from known examples a speculative axiomatization independent of the specification languages from previous sections. Note that we will also see a complete axiomatization in Section 4.5.1, although it diverges from Salomaa's style of axiomatization and uses an analog of the uniqueness axiom of Chapter 3 [Smo+20; BBK87].

**Star expressions and their operational semantics**

Fix an iterative branching theory $(S, \mathsf{EQ}, \mathsf{fp})$ and assume $S$ consists of only constants and binary operations. Its *star fragment* is the set[12] $StExp$ given by

$$StExp \ni e, e_1, e_2 ::= c \mid 1 \mid a \mid e_1 +_\sigma e_2 \mid e_1 e_2 \mid e^{(\sigma)}$$

where $a \in Act$, $c \in S_0$, and $\sigma \in S_2$.

The star fragment of an algebraic theory is a fragment of $Exp$ in the sense that star expressions can be thought of as shorthands for process terms, as we explain next. In this translation, we fix a distinguished variable $\underline{u} \in Var$, called the *unit*, which will denote successful termination, and we also fix a variable $v$ distinct from the unit, which will appear in the fixed-point. The translation $stp\colon StExp \to Exp$ is defined

$$stp(1) = \underline{u} \qquad stp(a) = a\underline{u} \qquad stp(e_1 +_\sigma e_2) = \sigma(stp(e_1), stp(e_2))$$
$$stp(e_1 e_2) = stp(e_1)[stp(e_2)/\underline{u}] \qquad stp(e^{(\sigma)}) = \mu v\ \sigma(stp(e)[v/\underline{u}], \underline{u})$$

Sequential composition of terms is associative and distributes over branching operations on the right-hand[13] side: For any $e_1, e_2, f \in StExp$, $(e_1 +_\sigma e_2)f$ and $e_1 f +_\sigma e_2 f$ translate to the same process term. For example, in the algebra of regular behaviours, this is representative of the intuitive right-distribution of sequential composition over nondeterministic choice. Similarly, the intuitively correct identities $1e = e = e1$ hold modulo translation, as well as the identity $ce = c$.[14]

The operational semantics for star expressions is given by an $L_M$-coalgebra

[12] Abstractly, $(StExp, \mathsf{ev}^*)$ is the initial $\Omega_M$-algebra where $\Omega_M = 1 + Act + S + \mathrm{Id}^2 + \{{}^{(\sigma)} \mid \sigma \in S_2\} \times \mathrm{Id}$.
[13] But not on the left-hand side! Observe the difference between the processes $a(b+c)$ and $ab+ac$.
[14] But not $ec = c$! See also the previous footnote.

$$\begin{aligned}
\ell(c) &= c & \ell(e_1 +_\sigma e_2) &= \sigma(\ell(e_1), \ell(e_2)) \\
\ell(1) &= \checkmark & \ell(ef) &= p(\ell(f), (a_1, e_1 f), \ldots, (a_n, e_n f)) \\
\ell(a) &= (a, 1) & \ell(e^{(\sigma)}) &= \mathsf{fp}\, \underline{u}\, \sigma\big(p(\underline{u}, (a_1, e_1 e^{(\sigma)}), \ldots, (a_n, e_n e^{(\sigma)})), \checkmark\big)
\end{aligned}$$

**Figure 4.4:** The structure map $\ell\colon StExp \to L_M StExp$. Here, $c$ is a constant of $S$, $\sigma$ is a binary operation of $S$, $a \in Act$, and $e, e_i \in StExp$. In the last two equations, $\ell(e) = p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n))$ for some $p \in S^*(\checkmark + Act \times StExp)$.

$(StExp, \ell)$ in Figure 4.4, where

$$L_M = M(\checkmark + Act \times \mathrm{Id}) \tag{4.12}$$

Abstractly, the operational interpretation $\ell(e)$ of a star expression $e$ is obtained by translating $e$ into a process term (also called $e$) and then identifying $\underline{u}$ with $\checkmark$ in $\varepsilon(e)$. While the notation is somewhat opaque at this level of generality, in specific instances the map $\ell$ amounts to a familiar transition structure.

*Example* 4.5.1. The star fragment of ACF from Example 4.1.11 and Example 4.2.7 coincides with GKAT, the algebra of programs introduced in [KT08] and studied further in [Smo+20; Sch+21]. Instantiating $StExp$ in this context reveals the syntax

$$e_i ::= 0 \mid 1 \mid p \mid e_1 +_A e_2 \mid e_1 e_2 \mid e^{(A)}$$

for $A \subseteq At$ and $p \in Act$, where $Act = \Sigma$ from Chapter 3. This is nearly the syntax of GKAT, the only difference being the presence of 1 and 0 instead of Boolean constants $b \in BExp$. This is merely cosmetic, as we can just as well define $b = 1 +_{\{\alpha \leq_{\mathsf{BA}} b\}} 0$. [15]

In this context, $M = (\bot + \mathrm{Id})^{At}$, and so $L_M \cong (2 + Act \times \mathrm{Id})^{At}$, which is the precise coalgebraic signature of the automaton models of GKAT expressions.

*Remark* 4.5.2. In Appendix A, I give back-and-forth translations between GKAT and the star fragment of ACF. These translations are easily seen to be $L_M$-coalgebra homomorphisms, with $M = (\bot + \mathrm{Id})^{At}$. Furthermore, up to the bisimulation GKAT axioms and the axioms for star expressions that will appear shortly, these back-and-forth translations are inverse to one another. It is in this sense that bisimulation GKAT is equivalent to the star fragment of ACF.

[15] Note that unlike the syntax currently used in the GKAT literature [Smo+20; Sch+21; KSS23; ZSS22], this syntax is finitary without having to quotient the guards by the axioms of Boolean algebra.

*Example* 4.5.3. The star fragment of APA from Example 4.1.12 and Example 4.2.8 gives a sort of "coin-flip" version of regular expressions, with an iteration operator for each $p \in [0,1]$. Instantiating *StExp* in this context reveals the syntax

$$e_i ::= 0 \mid 1 \mid a \mid e_1 \oplus_r e_2 \mid e_1 e_2 \mid e^{[r]}$$

for $r \in [0,1]$ and $a \in Act$. The process $e^{[r]}$ can be thought of as a generalized Bernoulli process that runs $e$ until it reaches ✓ and then flips a weighted coin to decide whether to start from the beginning of $e$ or to terminate successfully.

*Example* 4.5.4. A sort of combination of the previous two examples is *probabilistic guarded Kleene algebra with tests* (ProbGKAT) [Róż+23]. Fix a set of *output values* *Out* and atoms *At* and define

$$S = \{0\} + Out + [0,1] \times \mathrm{Id}^2 + 2^{At} \times \mathrm{Id}^2$$
$$\mathsf{PG} = \mathsf{CA} \cup \mathsf{GA} \cup \{x \oplus_r (y +_b z) = (x \oplus_r z) +_b (y \oplus_r z) \mid b \subseteq At\}$$

The monad presented by $(S, \mathsf{PG})$ is the set of $At$-indexed subdistributions over $Out + \mathrm{Id}$, $(\mathcal{D}(\bot + Out + \mathrm{Id})^{At}, \lambda\alpha.\delta_{(-)}, \lambda\alpha.\Delta_\alpha)$. That is, given a set $X$,

$$(\lambda\alpha.\delta_{(-)})_X(x)(\alpha) = \delta_x$$
$$(\lambda\alpha.\Delta_\alpha)_X(\Phi)(\alpha)(\xi) = \sum_\theta \Phi(\alpha)(\theta) \cdot \theta(\alpha)(\xi)$$

This can be turned into an iterative branching theory by adding a reduction operator whose action on the monad is

$$(\mathsf{fp}\, x\ \theta)(\alpha)(\xi) = \begin{cases} 0 & \xi = u \text{ or } \theta(\alpha)(u) = 1 \\ \frac{\theta(\alpha)(\xi)}{1-\theta(\alpha)(x)} & \text{otherwise} \end{cases}$$

The resulting syntax is

$$e, e_1, e_2 ::= 0 \mid 1 \mid c \in Out \mid p \in Act \mid e_1 +_A e_2 \mid e_1 \oplus_r e_2 \mid ef \mid e^{(A)} \mid e^{[r]}$$

for $A \subseteq At$ and $r \in [0,1]$, which is equivalent to the one found in [Róż+23] so long as we allow for the shorthand $b = 1 +_{\{\alpha \leq_{\mathsf{BA}} b\}} 0$ like in Example 4.5.1.

**Axioms for star expressions**

We now provide a candidate axiomatization for the star fragment while leaving the question of completeness open. Say that a star expression $e$ is *guarded* if the unit is guarded in $e$ as an expression in $Exp$. We define $\mathsf{EQ}^*$ to be the theory consisting of $\mathsf{EQ}$, the axioms

$$(\mathsf{EQ}^*1)\quad 1e = e1 = e \qquad (\mathsf{EQ}^*3)\quad e_1(e_2e_3) = (e_1e_2)e_3$$
$$(\mathsf{EQ}^*2)\quad ce = c \qquad (\mathsf{EQ}^*4)\quad e_1^{(\sigma)}e_2 = e_1e_1^{(\sigma)}e_2 +_\sigma e_2$$

where $c \in S_0$, $\sigma \in S_2$, and $p(x,\vec{y}) \in S^*X$, and the inference rules

$$(\mathsf{EQ}^*5)\quad \frac{(\forall i \le n)\ g_i \text{ is guarded}}{p(1,\vec{g})^{(\sigma)} = \mathsf{fp}\,\underline{u}\ (p(\underline{u},\vec{g}p(1,\vec{g})^{(\sigma)}) +_\sigma 1)}$$

$$(\mathsf{EQ}^*6)\quad \frac{g = eg +_\sigma f \qquad e \text{ is guarded}}{g = e^{(\sigma)}f}$$

The resulting system $\mathsf{EQ}^*$ is equivalent to bisimulation GKAT (Figure 3.5) when instantiated to guarded algebras (verifying this claim is the subject of Appendix A), equivalent to Milner's axiomatization of regular expressions modulo bisimilarity (Section 2.1) when instantiated to semilattices with bottom (see Example 4.1.7), and equivalent to the axioms of ProbGKAT [Róż+23] when instantiated to the iterative branching theory in Example 4.5.4.[16]

Write $(Z,\zeta)$ for the final $L_M$-coalgebra, and $!_\beta\colon (X,\beta) \to (Z,\zeta)$ for the unique coalgebra homomorphism from an $L_M$-coalgebra $(X,\beta)$ into $(Z,\zeta)$. The operational semantics of star expressions is given by the behaviour map $!_\ell\colon (StExp,\ell) \to (Z,\zeta)$.

**Theorem 4.5.5** (Soundness)**.** *Let $e,f \in StExp$. If $\mathsf{EQ}^* \vdash e = f$, then $!_\ell(e) = !_\ell(f)$.*

*Proof.* We begin by observing that the natural inclusion $\mathrm{in}\colon L_M \hookrightarrow B_M$ witnesses the following: Given $L_M$-coalgebras $(X,\beta_X),(Y,\beta_Y)$, and states $x \in X$ and $y \in Y$, $x$ and $y$ are behaviourally equivalent if and only if they are behaviourally equivalent as states of $B_M$-coalgebras $\mathrm{in}_*(X,\beta_X)$ and $\mathrm{in}_*(Y,\beta_Y)$ respectively. It therefore suffices to check two things: (1) The translation map $stp\colon \mathrm{in}_*(StExp,\ell) \hookrightarrow (Exp,\varepsilon)$ is an injective $B_M$-coalgebra homomorphism, and (2) that each axiom of $\mathsf{EQ}^*$ is taken to an equation

[16] This framework is actually the origin of ProbGKAT. ProbGKAT is studied in detail in [Róż+23], where decidability is also addressed.

provable in EFA by the translation map $stp$.

Statement (1) amounts to checking the equation

$$\varepsilon \circ stp(e) = \mathrm{in} \circ L_M(stp) \circ \ell(e) \tag{4.13}$$

This can be checked by induction on star expressions. The only interesting cases are the sequential composition and star cases. Assume (4.13) holds for subexpressions of $e$ and $f$, and let $\ell(e) = p(\checkmark, (a_1, g_1), \ldots, (a_n, g_n))$. Then

$$\mathrm{in} \circ L_M(stp) \circ \ell(e) = p(\underline{u}, (a_1, stp(g_1)), \ldots, (a_n, stp(g_n)))$$

Using this identity, we can deduce the following:

$$\begin{aligned}
\mathrm{in} \circ L_M(stp) \circ \ell(ef) &= \mathrm{in} \circ L_M(stp)(p(\ell(f), (a_1, g_1 f), \ldots, (a_n, g_n f))) \\
&= \mathrm{in}(p(L_M(stp) \circ \ell(f), (a_1, stp(g_1 f)), \ldots, (a_n, stp(g_n f)))) \\
&= p(\mathrm{in} \circ L_M(stp) \circ \ell(f), (a_1, stp(g_1 f)), \ldots, (a_n, stp(g_n f))) \\
&= p(\varepsilon \circ stp(f), (a_1, stp(g_1 f)), \ldots, (a_n, stp(g_n f))) \\
&= p(\varepsilon \circ stp(f), (a_1, stp(g_1)[stp(f)/\underline{u}]), \ldots, (a_n, stp(g_n)[stp(f)/\underline{u}])) \\
&= \varepsilon(stp(e)[stp(f)/\underline{u}]) \\
&= \varepsilon \circ stp(ef)
\end{aligned}$$

For the star case,

$$\begin{aligned}
&\mathrm{in} \circ L_M(stp) \circ \ell(e^{(\sigma)}) \\
&= \mathrm{in} \circ L_M(stp)(\mathsf{fp}\ \underline{u}\ \sigma(p(\underline{u}, (a_1, e_1 e^{(\sigma)}), \ldots, (a_n, e_n e^{(\sigma)})), \checkmark)) \\
&= \mathrm{in}\big(\mathsf{fp}\ \underline{u}\ \sigma(p(\underline{u}, (a_1, stp(e_1 e^{(\sigma)})), \ldots, (a_n, stp(e_n e^{(\sigma)}))), \checkmark)\big) \\
&= \mathrm{in}\big(\mathsf{fp}\ v\ \sigma(p(v, (a_1, stp(e_1)[stp(e^{(\sigma)})/\underline{u}]), \ldots, (a_n, stp(e_n)[stp(e^{(\sigma)})/\underline{u}])), \checkmark)\big) \\
&= \mathsf{fp}\ v\ \sigma(p(v, (a_1, stp(e_1)[stp(e^{(\sigma)})/\underline{u}]), \ldots, (a_n, stp(e_n)[stp(e^{(\sigma)})/\underline{u}])), \underline{u}) \\
&= \mathsf{fp}\ v\ \sigma(p(v, (a_1, stp(e_1)), \ldots, (a_n, stp(e_n))), \underline{u})[stp(e^{(\sigma)})//\underline{u}] \\
&= \mathsf{fp}\ v\ \sigma(p(v, (a_1, stp(e_1)), \ldots, (a_n, stp(e_n))), \underline{u})[\mu v\ \sigma(stp(e)[v/\underline{u}], \underline{u})//\underline{u}] \\
&= \mathsf{fp}\ v\ \varepsilon(\sigma(stp(e)[v/\underline{u}], \underline{u}))[\mu v\ \sigma(stp(e)[v/\underline{u}], \underline{u})//\underline{u}] \\
&= \varepsilon(\mu v\ \sigma(stp(e)[v/\underline{u}], \underline{u}))
\end{aligned}$$

$$= \varepsilon \circ stp(e^{(\sigma)})$$

This concludes the proof of (1). To see that (2) holds, check each axiom individually. The most interesting cases are (EQ*3)-(EQ*6).

Actually, (EQ*3) is a consequence of substitution being associative. That is, for $e_1, e_2, e_3 \in Exp$, we always have

$$e_1[e_2[e_3/\underline{u}]/\underline{u}] = e_1[e_2/\underline{u}][e_3/\underline{u}]$$

where equality here is explicit equality of terms. The rule (EQ*4) is not quite as straightforward. Let $e_1, e_2 \in StExp$ now, and compute

$$\begin{aligned}
\mathsf{EFA} \vdash stp(e_1^{(\sigma)} e_2) &= stp(e_1^{(\sigma)})[stp(e_2)/\underline{u}] \\
&= \mu v\ \sigma(stp(e_1)[v/\underline{u}], \underline{u})[stp(e_2)/\underline{u}] \\
&= \mu v\ \sigma(stp(e_1)[v/\underline{u}], stp(e_2)) \\
&= \sigma(stp(e_1)[v/\underline{u}], stp(e_2))[\mu v\ \sigma(stp(e_1)[v/\underline{u}], stp(e_2))/v] \qquad \text{(R1)} \\
&= \sigma(stp(e_1)[v/\underline{u}][\mu v\ \sigma(stp(e_1)[v/\underline{u}], stp(e_2))/v], stp(e_2)) \\
&= \sigma(stp(e_1)[\mu v\ \sigma(stp(e_1)[v/\underline{u}], stp(e_2))/\underline{u}], stp(e_2)) \\
&= \sigma(stp(e_1)[stp(e_1^{(\sigma)} e_2)/\underline{u}], stp(e_2)) \\
&= stp(e_1(e_1^{(\sigma)} e_2) +_\sigma e_2)
\end{aligned}$$

Next we check (EQ*5). If $g_1, \ldots, g_n$ are guarded star expressions, then

$$\begin{aligned}
\mathsf{EFA} \vdash stp(p(1, \vec{g})^{(\sigma)}) &= \mu v\ \sigma(p(v, stp(\vec{g})[v/\underline{u}]), \underline{u}) \\
&= \mu v\ (\mathsf{fp}\ v\ \sigma(p(v, stp(\vec{g})[v/\underline{u}]), \underline{u})) \qquad \text{(R3)} \\
&= \mathsf{fp}\ v\ \sigma(p(v, stp(\vec{g})[\mu v\ \sigma(p(v, stp(\vec{g})[v/\underline{u}]), \underline{u})/\underline{u}]), \underline{u}) \\
&= \mathsf{fp}\ v\ \sigma(p(v, stp(\vec{g})[stp(p(v, \vec{g}))/\underline{u}]), \underline{u}) \\
&= \mathsf{fp}\ v\ \sigma(p(v, stp(\vec{g} p(v, \vec{g}))), \underline{u}) \\
&= stp(\mathsf{fp}\ v\ p(v, \vec{g} p(v, \vec{g})) +_\sigma 1)
\end{aligned}$$

Finally, (EQ*6) translates directly into (R4). □

Grabmayer's Theorem 2.1.9 [Gra22] implies that the theory of semilattices $\mathsf{SL}^*$ is furthermore complete for behavioural equivalence, and the research literature appears to be closing in fast on a proof of the completeness of bisimulation GKAT (here, $\mathsf{GA}^*$). We are also confident that a completeness result can be obtained in other instances of the framework for the axiomatization we have suggested above.

**Question 1.** Let $(S, \mathsf{EQ}, \mathsf{fp})$ be an iterative branching theory. If $e$ and $f$ are behaviourally equivalent star expressions, is it true that $\mathsf{EQ}^* \vdash e = f$?

Restricting to specifically the *one-free star expressions* (like in Chapters 2 and 3), this question is even more potent, given the positive results covered in previous chapters. In Chapter 2, we saw that the one-free fragment of $\mathsf{SL}^*$ is a complete axiomatization of one-free regular expressions modulo bisimilarity (Theorem 2.4.13), and in Chapter 3 we saw that the one-free (i.e., skip-free) adaptation of $\mathsf{GA}^*$ is a complete axiomatization of skip-free bisimulation GKAT (Theorem 3.2.18).

**Question 2.** Let $(S, \mathsf{EQ}, \mathsf{fp})$ be an iterative branching theory. Define the *one-free star expressions* $StExp^-$ by the grammar

$$e_1, e_2 ::= a \in Act \mid c \in S_0 \mid e_1 e_2 \mid e_1 + _\sigma e_2 \mid e_1 \,{}^{(\sigma)}\, e_2$$

where $S_0$ contains the constants of $S$ and with $\sigma \in S_2$. We obtain the axiomatization $\mathsf{EQ}^*{}_0$ from $\mathsf{EQ}^*$ by deleting ($\mathsf{EQ}^*1$) and ($\mathsf{EQ}^*5$). If $e, f \in StExp^-$ are behaviourally equivalent, is it true that $\mathsf{EQ}^*{}_0 \vdash e = f$?

### 4.5.1 One Last Completeness Theorem

In this final technical subsection, we provide partial answers to **Question 1** and **Question 2** above. We prove a completeness theorem for star fragments with an additional axiom scheme that guarantees at most one solution to every finite system of equations, generalizing the *uniqueness axiom* in Chapter 3. Again, our completeness theorem will follow the structure of Lemma 2.5.3, which requires that we show that certain systems of equations are solvable. We make use of the following more succinct depiction of systems of equations, akin to the *flat systems of equations* from the iteration theories literature [BE76; BM96].

Given a set $X$, define $StExp(X)$ to be the set of terms given by the grammar

$$StExp(X) \ni t_1, t_2, t ::= c \in S_1 \mid \xi_1 +_\sigma \xi_2 \qquad \xi_1, \xi_2 ::= ex \mid t$$

where $e \in StExp$ and $x \in X$.

**Definition 4.5.6.** A *left-affine system of equations* is a function $\alpha\colon X \to StExp(X)$. We call $\alpha$ *finite* if $X$ is finite, and we call $\alpha$ *Salomaa* if for every $x \in X$ and every $fy$ that appears in $\alpha(x)$, $f$ is a guarded star expression. Given an equivalence relation $\approx$ on $StExp$, A $\approx$*-solution* to $\alpha$ is a function $\varphi\colon X \to StExp$ such that for every $x \in X$, $\varphi(x) \approx \varphi^\# \circ \alpha(x)$, where $\varphi^\#\colon StExp(X) \to StExp$ is defined

$$\varphi^\#(c) = c \qquad \varphi^\#(ex) = e\varphi(x) \qquad \varphi^\#(t_1 +_\sigma t_2) = \varphi^\#(t_1) +_\sigma \varphi^\#(t_2)$$

for any $c \in StExp$, $e \in StExp$, $x \in X$. Two $\approx$-solutions $\varphi_1, \varphi_2$ to $\alpha$ are $\approx$*-equivalent*, written $\varphi_1 \approx \varphi_2$, if $\varphi_1(x) \approx \varphi_2(x)$ for all $x \in X$.

**Definition 4.5.7.** The *generalized uniqueness axiom* is the axiom scheme that states that up to provable equivalence, there is at most one solution to every finite Salomaa left-affine system. We write $\equiv^+$ for the smallest congruence that contains the axioms of $\mathsf{EQ}^*$ and satisfies the generalized uniqueness axiom.

This is, in fact, sound with respect to behavioural equivalence.

**Lemma 4.5.8.** *The congruence $\equiv^+$ is the kernel of a coalgebra homomorphism.*

*Proof.* We need to fill in the diagram below

$$\begin{array}{ccc} StExp & \xrightarrow{[-]_{\equiv^+}} & StExp/{\equiv^+} \\ {\scriptstyle \ell}\downarrow & & \downarrow{\scriptstyle \bar{\ell}} \\ L_M StExp & \xrightarrow[L_M([-]_{\equiv^+})]{} & L_M(StExp/{\equiv^+}) \end{array} \tag{4.14}$$

Like in the proof of Lemma 4.4.3, ensuring that such a $\bar{\ell}$ exists amounts to showing that whenever $e \equiv^+ f$,

$$L_M([-]_{\equiv^+}) \circ \ell(e) = L_M([-]_{\equiv^+}) \circ \ell(f) \tag{4.15}$$

We proceed by induction on the proof of $e \equiv^+ f$.

In the base case, we have to check each of the equational axioms of $\mathsf{EQ}^*$. For $(p(\vec{x}), q(\vec{x})) \in \mathsf{EQ}$ with $\vec{x} = (x_1, \ldots, x_n)$, we have $\ell(p(\vec{e})) = \ell(q(\vec{e}))$ for any $\vec{e} \in StExp^n$. For ($\mathsf{EQ}^*1$), we have

$$\ell(1e) = p(\ell(e), (a_1, e_1), \ldots, (a_n e_n)) = \ell(e)$$

where $\ell(1) = p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n)) = \checkmark$. On the other hand, if $\ell(e) = p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n))$, then

$$\begin{aligned} L_M([-]_{\equiv^+}) \circ \ell(e1) &= p(\ell(1), (a_1, [e_1 1]_{\equiv^+}), \ldots, (a_n, [e_n 1]_{\equiv^+})) \\ &= p(\checkmark, (a_1, [e_1 1]_{\equiv^+}), \ldots, (a_n, [e_n 1]_{\equiv^+})) \\ &= p(\checkmark, (a_1, [e_1]_{\equiv^+}), \ldots, (a_n, [e_n]_{\equiv^+})) \\ &= L_M([-]_{\equiv^+}) \circ \ell(e) \end{aligned}$$

Applying $L_M([-]_{\equiv^+})$, this is equivalent to $\ell(e)$. For ($\mathsf{EQ}^*2$), we know that $\ell(c) = c$, so $\ell(ce) = c = \ell(c)$. For ($\mathsf{EQ}^*3$), let $\ell(e_1) = p(\checkmark, (a_1, e_{11}), \ldots, (a_n, e_{1n}))$ and $\ell(e_2) = q(\checkmark, (b_1, e_{21}), \ldots, (b_m, e_{2m}))$. Then

$$\begin{aligned} L_M([-]_{\equiv^+}) \circ \ell(e_1(e_2e_3)) &= p(\ell(e_2e_3), (a_1, [e_{11}(e_2e_3)]^+_{\equiv}), \ldots, (a_n, [e_{1n}(e_2e_3)]_{\equiv^+})) \\ &= p(q(\ell(e_3), (b_1, [e_{21}e_3]_{\equiv^+}), \ldots, (b_m, [e_{2m}e_3]_{\equiv^+})), \\ &\qquad (a_1, [e_{11}(e_2e_3)]_{\equiv^+}), \ldots, (a_n, [e_{1n}(e_2e_3)]_{\equiv^+})) \\ &= p(q(\ell(e_3), (b_1, [e_{21}e_3]_{\equiv^+}), \ldots, (b_m, [e_{2m}e_3]_{\equiv^+})), \\ &\qquad (a_1, [(e_{11}e_2)e_3]_{\equiv^+}), \ldots, (a_n, [(e_{1n}e_2)e_3]_{\equiv^+})) \quad (\mathsf{EQ}^*3) \\ &= L_M([-]_{\equiv^+}) \circ \ell((e_1e_2)e_3) \end{aligned}$$

For ($\mathsf{EQ}^*4$), we compute

$$\begin{aligned} \ell(e_1{}^{(\sigma)}e_2) &= \mathsf{fp}\,\underline{u}\;\sigma(p(\underline{u}, (a_1, (e_{11}e_1{}^{(\sigma)})e_2), \ldots, (a_n, (e_{1n}e_1{}^{(\sigma)})e_2)), \ell(e_2)) \\ &= \sigma(p(\mathsf{fp}\,\underline{u}\;\sigma(p(\underline{u}, (a_1, (e_{11}e_1{}^{(\sigma)})e_2), \ldots, (a_n, (e_{1n}e_1{}^{(\sigma)})e_2)), \ell(e_2)), \\ &\qquad (a_1, (e_{11}e_1{}^{(\sigma)})e_2), \ldots, (a_n, (e_{1n}e_1{}^{(\sigma)})e_2)), \ell(e_2)) \\ &= \sigma(p(\ell(e_1{}^{(\sigma)}e_2), (a_1, (e_{11}e_1{}^{(\sigma)})e_2), \ldots, (a_n, (e_{1n}e_1{}^{(\sigma)})e_2)), \ell(e_2)) \\ &= \ell((e_1e_1{}^{(\sigma)})e_2 +_\sigma e_2) \end{aligned}$$

For (EQ*5), assume $g_1,\ldots,g_n$ are guarded, and let $\ell(g_i) = q_i((a_{i1},g_{i1}),\ldots,(a_{im_i},g_{im_i}))$. Then compute

$$\begin{aligned}\ell(p(1,\vec{g})^{(\sigma)}) &= \mathsf{fp}\,\underline{u}\;\sigma(p(\underline{u},q_1((a_{1j},g_{1j}p(1,\vec{g})^{(\sigma)})),\ldots,q_n((a_{nj},g_{nj}p(1,\vec{g})^{(\sigma)}))),\checkmark)\\ &= \mathsf{fp}\,\underline{u}\;\sigma(p(\underline{u},\ell(g_1p(1,\vec{g})^{(\sigma)}),\ldots,\ell(g_np(1,\vec{g})^{(\sigma)})),\checkmark)\\ &= \ell(\mathsf{fp}\,\underline{u}\;\sigma(p(\underline{u},g_1p(1,\vec{g})^{(\sigma)},\ldots,g_np(1,\vec{g})^{(\sigma)}),\checkmark))\end{aligned}$$

since $\ell$ is an $S$-algebra homomorphism.

We skip (EQ*6) and prove that if the final step in the proof of $e \equiv^+ f$ is an application of the generalized uniqueness axiom, then (4.15) holds. The generalized uniqueness axiom subsumes (EQ*6), since (EQ*6) is the generalized uniqueness axiom in the special case where the set of indeterminates in the Salomaa left-affine system has only one element.

Finally, suppose that $\varphi_1,\varphi_2$ are the $\equiv^+$-solutions to a finite Salomaa left-affine system of equations $\alpha\colon X \to StExp(X)$ that witnesses $e \equiv^+ f$. The induction hypothesis tells us that for any $x \in X$,

$$L_M([-]_{\equiv^+}) \circ \ell(\varphi_i(x)) = L_M([-]_{\equiv^+}) \circ \ell(\varphi_i^{\#}(\alpha(x)))$$

where $i \in \{1,2\}$. We need to show that this implies $L_M([-]_{\equiv^+}) \circ \ell \circ \varphi_1 = L_M([-]_{\equiv^+}) \circ \ell \circ \varphi_2$. To this end, let $x \in X$ and $\alpha(x) = p(e_1x_1,\ldots,e_nx_n,f)$ with $e_1,\ldots,e_n$ guarded. Furthermore, let $\ell(e_i) = q_i((a_{i1},e_{i1}),\ldots,(a_{im_i},e_{im_i}))$. Then

$$\begin{aligned}&L_M([-]_{\equiv^+}) \circ \ell(\varphi_1(x))\\ &= L_M([-]_{\equiv^+}) \circ \ell(\varphi_1^{\#}(\alpha(x)))\\ &= L_M([-]_{\equiv^+}) \circ \ell(p(e_1\varphi_1(x_1),\ldots,e_n\varphi_1(x_n),f))\\ &= p(L_M([-]_{\equiv^+}) \circ \ell(e_1\varphi_1(x_1)),\ldots,L_M([-]_{\equiv^+}) \circ \ell(e_n\varphi_1(x_n)),\\ &\qquad L_M([-]_{\equiv^+}) \circ \ell(f))\\ &= p(q_1((a_{11},[e_{11}\varphi_1(x_1)]_{\equiv^+}),\ldots,(a_{1m_1},[e_{1m_1}\varphi_1(x_n)]_{\equiv^+})),\ldots,\\ &\qquad q_n((a_{n1},[e_{n1}\varphi_1(x_1)]_{\equiv^+}),\ldots,(a_{nm_n},[e_{nm_n}\varphi_1(x_n)]_{\equiv^+})),\\ &\qquad L_M([-]_{\equiv^+}) \circ \ell(f))\\ &= p(q_1((a_{11},[e_{11}\varphi_2(x_1)]_{\equiv^+}),\ldots,(a_{1m_1},[e_{1m_1}\varphi_2(x_n)]_{\equiv^+})),\ldots,\end{aligned}$$

$$
\begin{aligned}
& q_n((a_{n1}, [e_{n1}\varphi_2(x_1)]_{\equiv^+}), \ldots, (a_{nm_n}, [e_{nm_n}\varphi_2(x_n)]_{\equiv^+})), \\
& L_M([-]_{\equiv^+}) \circ \ell(f)) && \text{(UA)} \\
= {} & L_M([-]_{\equiv^+}) \circ \ell(\varphi_2^{\#}(\alpha(x))) \\
= {} & L_M([-]_{\equiv^+}) \circ \ell(\varphi_2(x))
\end{aligned}
$$

as desired. This concludes the proof of the lemma. □

Similar to how Lemma 4.4.3 implies the soundness of our axiomatization of effectful process algebra, Lemma 4.5.8 immediately implies the soundness of EQ$^*$.

**Theorem 4.5.9** (Soundness)**.** *For $e, f \in StExp$, if $e \equiv^+ f$, then $!_\ell(e) = !_\ell(f)$.*

**Completeness**

To establish the completeness of $\equiv^+$, we are going to follow the completeness proof of bisimulation GKAT with the uniqueness axiom (Section 3.1). The crux of the completeness proof is a one-to-one correspondence between coalgebra homomorphisms into $(StExp/\equiv^+, [\ell]_{\equiv^+})$ and solutions to left-affine systems of equations specified by $L_M$-coalgebras. By the generalized uniqueness axiom, this tells us that $L_M$-coalgebras admit at most one homomorphism into $(StExp/\equiv^+, [\ell]_{\equiv^+})$.

Up to EQ, every finite $L_M$-coalgebra $(X, \delta)$ corresponds to a Salomaa left-affine system of equations $\hat{\delta}\colon X \to StExp(X)$ obtained by letting $\hat{\delta}(x)$ be a term representing $\delta(x)$ in $S^*(1 + Act \times X) \subseteq StExp(X)$. Up to EQ, $\hat{\delta}$ is uniquely defined by this choice. The following lemma is the one-to-one correspondence we need.

**Lemma 4.5.10.** *Let $(X, \delta)$ be a finite $\mathcal{L}_M$-coalgebra. Then a function $\varphi\colon X \to StExp$ is a $\equiv^+$-solution to $(X, \hat{\delta})$ if and only if $[-]_{\equiv^+} \circ \varphi$ is a coalgebra homomorphism.*

Like before, we need a version of the fundamental theorem in the proof of Lemma 4.5.10. This can be stated as follows.

**Lemma 4.5.11.** *Let $e \in StExp$ and $\ell(e) = p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n))$. Then we have the provable equivalence $e \equiv^+ p(1, a_1 e_1, \ldots a_n e_n)$.*

Of course, here we are taking $\sigma(x, y)$ and $x +_\sigma y$ to be synonymous.

*Proof.* By induction on $e$. In the base cases, we have $\ell(c) = c$ and $\ell(1) = \checkmark$, both trivial. In the induction hypothesis, assume the result for $e, f$ and let

$\ell(e) = p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n))$ and $\ell(f) = q(\checkmark, (b_1, f_1), \ldots, (b_n, f_n))$. Then by the induction hypothesis,

$$\begin{aligned} e +_\sigma f &\equiv^+ p(1, a_1 e_1, \ldots, a_n e_n) +_\sigma q(1, b_1 f_1, \ldots, b_n f_n) \\ \ell(e +_\sigma f) &= \sigma(p(\checkmark, (a_1, e_1), \ldots, (a_n, e_n)), q(\checkmark, (b_1, f_1), \ldots, (b_n, f_n))) \end{aligned}$$

and in the sequential composition case,

$$\begin{aligned} ef &\equiv^+ p(1, a_1 e_1, \ldots, a_n e_n) f \\ &\equiv^+ p(f, a_1 e_1 f, \ldots, a_n e_n f) \\ &\equiv^+ p(q(1, b_1 f_1, \ldots, b_n f_n), a_1 e_1 f, \ldots, a_n e_n f) \\ \ell(ef) &= p(\ell(f), (a_1, e_1 f), \ldots, (a_n, e_n f)) \\ &= p(q(\checkmark, (b_1, f_1), \ldots, (b_n, f_n)), (a_1, e_1 f), \ldots, (a_n, e_n f)) \end{aligned}$$

For the star case, we first consider the case where $e$ is guarded. Using (EQ*4),

$$\begin{aligned} e^{(\sigma)} &\equiv^+ e e^{(\sigma)} +_\sigma 1 \\ &\equiv^+ p(a_1 e_1, \ldots, a_n e_n) e^{(\sigma)} +_\sigma 1 \\ &\equiv^+ p(a_1 e_1 e^{(\sigma)}, \ldots, a_n e_n e^{(\sigma)}) +_\sigma 1 \\ \ell(e^{(\sigma)}) &= \mathsf{fp}\, \underline{u}\; p((a_1, e_1 e^{(\sigma)}), \ldots, (a_n, e_n e^{(\sigma)})) \\ &= p((a_1, e_1 e^{(\sigma)}), \ldots, (a_n, e_n e^{(\sigma)})) \end{aligned}$$

In case $e$ is not guarded, we need to find a guarded $f \in \mathit{StExp}$ such that $e^{(\sigma)} \equiv^+ f^{(\sigma)}$. We can find such an $f$ by writing $e$ in the form $p(1, \vec{g})$ for some guarded $g_1, \ldots, g_n$ and some $S$-term $p$, and then using (EQ*5) to show that the desired $f$ is $\mathsf{fp}\, \underline{u}\; p(\underline{u}, \vec{g})$. This term $p$ and tuple of guarded expressions $\vec{g}$ can be constructed by induction on $e$. □

*Proof of Lemma 4.5.10.* Let $(X, \delta)$ be a finite $L_M$-coalgebra, and let $\varphi \colon X \to \mathit{StExp}$. Observe that if $\delta(x) = p(\checkmark, (a_1, x_1), \ldots, (a_n, x_n))$ and $\ell(\varphi(x)) = q(1, (b_1, f_1), \ldots, (b_m, f_m))$, then

$$\begin{aligned} L_M([-]_{\equiv^+} \circ \varphi) \circ \delta(x) &= L_M([-]_{\equiv^+} \circ \varphi)(p(\checkmark, (a_1, x_1), \ldots, (a_n, x_n))) \\ &= p(\checkmark, (a_1, [\varphi(x_1)]_{\equiv^+}), \ldots, (a_n, [\varphi(x_n)]_{\equiv^+})) \end{aligned}$$

$$\begin{aligned}\bar{\ell} \circ [-]_{\equiv^+} \circ \varphi(x) &= L([-]_{\equiv^+}) \circ \ell \circ \varphi(x) \\ &= q(\checkmark, (b_1, [f_1]_{\equiv^+}), \ldots, (b_m, [f_m]_{\equiv^+}))\end{aligned}$$

Observe that by the fundamental theorem, $\varphi(x) \equiv^+ q(1, b_1 f_1, \ldots, b_m f_m)$. So, if $[-]_{\equiv^+} \circ \varphi$ is a homomorphism, then $p(\checkmark, (a_1, [\varphi(x_1)]_{\equiv^+}), \ldots, (a_n, [\varphi(x_n)]_{\equiv^+})) = q(\checkmark, (b_1, [f_1]_{\equiv^+}), \ldots, (b_m, [f_m]_{\equiv^+}))$, which in particular means that

$$p(1, a_1\varphi(x_1), \ldots, a_n\varphi(x_n)) \equiv^+ q(1, b_1 f_1, \ldots, b_m f_m) \equiv^+ \varphi(x)$$

Since $x$ is arbitrary, $\varphi$ is a $\equiv^+$-solution. Conversely, suppose $\varphi$ is a $\equiv^+$-solution. Then $\varphi(x) \equiv^+ p(1, a_1\varphi(x_1), \ldots, a_n\varphi(x_n))$, or equivalently, $[-]_{\equiv^+} \circ \varphi(x) = p(1, [a_1\varphi(x_1)]_{\equiv^+}, \ldots, [a_n\varphi(x_n)]_{\equiv^+})$. Applying $\bar{\ell}$ to both sides gives

$$\bar{\ell} \circ [-]_{\equiv^+} \circ \varphi(x) = p(1, (a_1, [\varphi(x_1)]_{\equiv^+}), \ldots, (a_n, [\varphi(x_n)]_{\equiv^+})) = L_M([-]_{\equiv^+} \circ \varphi) \circ \delta(x) \quad \square$$

Under mild conditions on the algebraic theory $(S, \mathsf{EQ})$, the generalized uniqueness axiom allows for a complete axiomatization of behavioural equivalence.

**Theorem 4.5.12** (Completeness)**.** *Suppose $M$ preserves weak pullbacks, and let $e$ and $f$ be star expressions. If $!_\ell(e) = !_\ell(f)$, then $e \equiv^+ f$.*

*Proof.* Suppose $!_\ell(e) = !_\ell(f)$. Recall that when $B$ is a functor that preserves weak pullbacks, two states of a $B$-coalgebra are behaviourally equivalent if and only if they are bisimilar. The functor $\checkmark + Act \times (-)$ preserves weak pullbacks, because it is polynomial [Rut00]. We have also assumed that $M$ preserves weak pullbacks, and the composition of two weak pullback-preserving functors preserves weak pullbacks. It follows that $L_M = M(\checkmark + Act \times (-))$ preserves weak pullbacks. Thus, $e \leftrightarrows f$.

Let $(R, \delta_R)$ be a bisimulation on $\langle e \rangle \times \langle f \rangle$ that witnesses $e \leftrightarrows f$. Then $R$ is finite, and so by the generalized uniqueness axiom, $\hat{\delta}_R \colon R \to StExp$ has at most one solution. Let $\pi_1, \pi_2$ be the projections of $R$ onto $\langle e \rangle$ and $\langle f \rangle$ respectively. Then $[-]_{\approx^+} \circ \pi_1, [-]_{\approx^+} \circ \pi_2$ are homomorphisms, so $\pi_1, \pi_2$ are solutions to $\hat{\delta}_R$. By the generalized uniqueness axiom, $\pi_1 \equiv^+ \pi_2$, so $e \equiv^+ f$. $\square$

## 4.6 Related Work

Our framework can be seen as a generalization of Milner's ARB [Mil84] that reaches beyond nondeterministic choice and covers other process algebras already identified in the literature. For example, instantiating our framework in the theory of pointed convex algebras produces the algebra we have called APA (see Example 4.2.8), which appears to be very similar to the process calculus introduced by Stark and Smolka [SS00]. Instantiating our framework in the theory CS$\top$ of convex semilattices with top (see Example 4.1.14) produces a calculus that appears to be very similar to the calculus of Mislove, Ouaknine, and Worrell [MOW03]. In fact, I conjecture that APA is equivalent to Stark and Smolka's calculus (in the sense that the same equations are provable) and that the calculus for CS$\top$ is equivalent to (the equational part of) Mislove et al.'s calculus. More work needs to be done to verify these conjectures. The calculus obtained from the theory of pointed convex semilattices (Example 4.2.9) appears to be new.

### Algebraic effects

Throughout the thesis, I have made use of the terms *effect* and *branching*. Both terms are used in related literature. I have chosen to use them slightly differently.

The term *effect* comes from Plotkin and Power's *algebraic effects* [PP02], which are a refinement of Moggi's *notions of computation* [Mog89; Mog91]. Roughly, an algebraic effect is a computation type captured by a monad on a monoidal category that is presented by an algebraic theory. Examples of algebraic effects include nondeterminism, if-then-else, probabilities, exceptions, and input/output. Algebraic effects are specified syntactically using algebraic operations, the same way our effects are specified algebraically. An algebraic presentation of a monad $M$ furthermore induces a canonical *strength* on the monad, a natural transformation of the form $X \otimes MY \Rightarrow M(X \otimes Y)$ that satisfies certain coherence conditions, where $\otimes$ is a given monoidal product on the underlying category. If $\otimes$ is the usual Cartesian product $\times$ and $p(\vec{y})$ is a term representing an element of $MY$, the canonical strength can roughly be though of as the map

$$(x, p(y_1, \ldots, y_n)) \longmapsto p((x, y_1), \cdots, (x, y_n))$$

In the literature, the term *effect* typically refers to a monad equipped with a strength [Koc70; PP02]. Every of the monads we study in the thesis (in fact, every monad on **Set**) admits a strength, so our usage of the word *effect* is not wholly inconsistent with its use in the literature. We simply do not have any use for monad strengths at any point in the thesis.

The word *branching* also has a wider connotation, usually referring to *commutative* monads (see, for example [Cîr17]). Commutative monads are strong monads that capture algebraic effects with commutative algebraic operations, like nondeterminism (captured by the free semilattice construction) and probabilities (real numbers form a commutative semiring). We diverge significantly from the literature and use *branching* to refer to any algebraically structured set of outgoing state transitions instead, in reference to the tree-like structure of program behaviours (see Section 4.3.1 for example). This should not introduce any confusion, as commutativity does not play a role in this thesis: The monads capturing nondeterminism and probabilities are commutative, but the monad capturing if-then-else (in Example 4.1.11) is not.

**Star expressions**

Star expressions for nondeterministic processes appeared in the work of Milner [Mil84] and can be thought of as a bisimulation-focused analogue of Kleene's regular expressions for NFAs. While the syntaxes of Milner's star expressions and Kleene's regular expressions are the same, there are several important differences between their interpretations: For example, sequential composition is interpreted as the variable substitution $ef := e[f/\underline{u}]$ in Milner's paper, which fails to distribute over $+$ on the left. A notable insight from [Mil84] is that, despite these differences, an iteration operator $(-)^*$ can be defined for Milner's star expressions that satisfies many of the same identities as the Kleene star.

Given a variable $v$ distinct from $\underline{u}$ (the unit), and a process term $e$ of ARB such that $\mathrm{fv}(e) \subseteq \{\underline{u}\}$, $e^* := \mu v\ (e[v/\underline{u}] + \underline{u})$ defines the iteration operator in Milner's star fragment of ARB. In Section 4.5, we generalized this construction of Milner for the more general process types that we considered in this paper. The axiomatization $\mathsf{EQ}^*$ is also inspired by Milner's (and Salomaa's [Sal66]) work. We expect a general completeness theorem for star fragments is a hard problem, as completeness in the instantiation to ARB was open for decades despite the extensive literature on the

subject [FZ94; Fok97; FZ97; BCG06; GF20; Gra22].

### Kleene coalgebra

There are clear parallels between our work, the thesis of Silva (titled *Kleene coalgebra*) [Sil10], and its generalization to other categories than **Set** [Mil10; BMS13]. In this line of work, a family of calculi is introduced that generalizes Milner's process calculus (what we have called ARB) and includes one-exit[17] versions of ARB, ACF, and APA (see Examples 4.1.7, 4.2.7 and 4.2.8). Our work in this chapter generalizes Milner's process calculus in a different direction: Our framework is parametric on a finitary monad on **Set**, whereas Silva's is centred around one particular theory (semilattices) and provides specification languages for coalgebras of general polynomial functors on **Set**. The coalgebraic signatures captured by effectful process algebra do not currently include all such functors.

Our results are also in the same vein as the work of Myers on coalgebraic expressions [Mye09]. Coalgebraic expressions generalize the calculi of [Sil10] to arbitrary finitary coalgebraic signatures on a variety of algebras, and furthermore have totally defined recursion operators similar to ours. However, the focus of the framework of coalgebraic expressions is on language semantics, achieved by lifting the coalgebraic signature to a variety. This distinguishes the framework from our approach, since we focus on bisimulation semantics.

### Iterative theories and Elgot monads

Finally, there is also a notable connection to the iterative theories of Elgot [Elg75], Bloom and Ésik [BE76; BÉ88], and Nelson [Nel83]. The notion of an iterative branching theory (Definition 4.1.9) is a direct adaptation of Adamék, Milius, and Velebil's notion of Elgot monad [AMV11] to the specific situation of recursive specifications in one variable, like $x = x \oplus_{\frac{1}{2}} y$ in the case of convex algebra. Moreover, Theorem 4.4.10 implies that effectful process calculi (that include arbitrary $\mu$ operators) are examples of iterative algebras.

The main difference between effectful process calculi and iteration theories is that effectful process calculi use single-variable fixed-point operators ($\mu v$) to resolve recursive specifications, instead of working with recursive specifications

[17] There is a constant 1 for successful termination, instead of the set of output variables we allow.

directly. The restriction to single-variable fixed-points allows for an unrestricted syntax and an algebraic workflow when reasoning about program equivalence. It is also worth noting that many of the desirable properties of (equational) iteration theories are consequences of the uniqueness of guarded fixed-points constructed with the single-variable fixed-point operators. For example, Theorem 4.4.10 is a version of Bekić's lemma [Bek84], which allows for least fixed-points of multivariate recursive specifications to be computed one variable at a time.

Another thing that sets the effectful process algebra framework apart from the iteration theories mentioned is that interesting fragments of effectful process calculi are easily identifiable by syntactic constructions. Star fragments and their one-free counterparts are notable examples. For contrast, to identify a particular fragment of an iterative algebra requires structural constraints on the systems of equations that can be solved in that fragment [Nel83]. This can be very difficult: One of the open problems posed by Milner in [Mil84] asks for a structural characterization of systems solvable in regular expressions modulo bisimilarity. One solution was given by Baeten, Corradini, and Grabmayer in [BCG07]. Another solution to this structural characterization problem is at the heart of Grabmayer's solution to Milner's completeness problem [Gra22], where Grabmayer gives an intricate characterization of the labelled transition systems that have unique solutions in the algebra of regular expressions modulo bisimilarity (these play a similar role to the LLEE-charts of [GF20], which we renamed to well-layered precharts in Chapter 2).

## Chapter 5

# Conclusions and Future Work

In this thesis, we proved a number of completeness theorems for coalgebraic process calculi. We used several proof techniques, including the local and global approaches that were introduced in Chapter 2 and a technique that originates in the work of Salomaa [Sal66]. We also saw several reductions of one completeness problem to another: In Chapter 3, for example, we reduced the completeness of skip-free bisimulation GKAT (propositional while programs without a skip command) to the completeness theorem of regular expressions modulo bisimilarity due to Grabmayer and Fokkink [GF20]. The structural characterization of completeness theorems given in Chapter 2 was the key ingredient of the main completeness proof in Chapter 4, where I gave a uniform sound and complete axiomatization of behavioural equivalence in effectful process calculi, generalizing several examples from the literature.

### Effectful process calculi

I call the process calculi introduced in Chapter 4 *effectful*, in reference to Plotkin and Power's *algebraic effects* [PP02][1], because they are parametrized by algebraic theories. Effectful process calculi present a new perspective on several examples from the literature. For example, skip-free GKAT expressions make up a fragment of an effectful process calculus called ACF, which is obtained from the theory of guarded algebra (Example 4.1.11). One-free regular expressions, on the other hand, make up a fragment of the effectful process calculus ARB, obtained from the theory of semilattices with bottom. Our reduction of skip-free GKAT to one-free regular expressions modulo bisimilarity was obtained from an embedding of guarded

[1] The broader usage of the word *effect* is discussed under the *Algebraic effects* heading in the related work section, Section 4.6.

algebras into semilattices. In other words, this reduction was the result of treating the computational effects in these two calculi as first-class objects.

**Completeness for star fragments**

Regular expressions and GKAT expressions make up a special kind of fragment of their respective effectful process calculi, what we have called a star fragment. In Section 4.5.1 of Chapter 4, we saw that under a mild condition on the computational effect, a complete axiomatization of bisimilarity in star fragments of effectful process calculi can be obtained from a powerful axiom scheme called the (generalized) uniqueness axiom. The two completeness theorems of GKAT in Chapter 3 can both be derived from this completeness result for star fragments.

One drawback of the uniqueness axiom for star fragments is that it is difficult to apply in equivalence proofs. In [Mil84], Milner proposes an axiomatization of regular expressions modulo bisimilarity that is equivalent to an instance of our general axiomatization without the uniqueness axiom, and asks if his proposed axiomatization is complete. It took 38 years to answer this question [Gra22], despite intensive research efforts.

A similar problem was posed by Smolka et al. in [Smo+20], who asked whether the axioms of GKAT are complete for language equivalence. As we saw in Chapter 3, the completeness of GKAT can be reduced to the completeness of bisimulation GKAT, an instance of the star fragment construction. The completeness problems for both GKAT and bisimulation GKAT remain open.

Without a doubt, establishing the completeness of our axiomatization of bisimilarity (without the uniqueness axiom) in a given star fragment presents a difficult problem. Only one instance of this general completeness problem has been solved (Milner's completeness problem), so there is much work left to do.

## 5.1 Future Work

To finish, I would like to discuss possible further directions and speculate on the future of each of the research programs addressed in the thesis.

### 5.1.1 Coequations

A theorem of Rutten [Rut00] says that a class of coalgebras is closed under subcoalgebras, coproducts, and homomorphic images if and only if it is a *covariety*, a class of

coalgebras presented by a coequation in some number of colours. This is the dual of Birkhoff's HSP-theorem [Bir35]. The number of colours needed to present a covariety is (dually) analogous to the number of free variables needed to write an equational presentation of a variety. The first two research directions I would like to discuss deal with coequations.

**Direction 1: Coequations and well-layeredness**

In the first research direction, I would like to see a concrete description of a coequation that presents the covariety of locally well-layered precharts. The final $L$-coalgebra is a cofree coalgebra in one colour (it is a *behavioural* covariety [DS21]), so an initial guess at the coequation presenting the class of locally well-layered precharts might be the set of behaviours exhibited by well-layered precharts. However, a covariety presented by a coequation in one colour is closed under bisimilarity [GS01], which we know from Figure 2.5 is not true for locally well-layered precharts.

This leads us to the following question: How many colours are needed to characterize the covariety of locally well-layered precharts? I conjecture that the answer is infinite (specifically $\omega$), because it is likely that the number of colours that need to appear in a specific prechart to ensure well-layeredness is proportional to the depth of the shallowest $\curvearrowright$-graph of its layering witnesses (see Definition 2.4.2).

**Direction 2: Coequations and completeness**

The practicality of coequations in the pursuit of completeness theorems is not well-understood. In Chapter 2, we saw that the completeness proof of Grabmayer and Fokkink implicitly begins with a description of (what amounts to) a covariety, the covariety of locally well-layered precharts, but no concrete description of the coequation presenting this covariety is known. It would be interesting to see an example of a completeness proof that starts with a description of a coequation[2] and uses this concrete description to establish unique solvability of coalgebras satisfying the coequation. Even an example of an existing completeness proof that can be shortened with the use of coequations would be encouraging here. This leads us to the following question: Are there examples of completeness proofs that could be made easier by directly approaching them with coequations?

[2] See [DS21] for examples of such descriptions.

### 5.1.2 Guarded Kleene Algebra with Tests

In Chapter 3, we showed that GKAT and bisimulation GKAT extended with the uniqueness axiom are complete for their respective semantics.

**Direction 3: Adapting the completeness proof of GKAT with UA**

The proof that GKAT with the uniqueness axiom is complete consists of a reduction to bisimulation GKAT, which is a key example of a coalgebraic completeness theorem. However, the completeness proof in Section 3.1 was not in the style suggested by the local or global approaches[3] of Chapter 2. Instead, it was in the style of Salomaa: pushing solutions forward along projections from a bisimulation rather than pulling them back from a cospan of homomorphisms. Interestingly, our characterization of completeness theorems (Lemma 2.5.3) tells us that a local/global approach is possible, but constructing such a proof by hand appears to be difficult.

By the uniqueness axiom, if a GKAT automaton admits a solution, then it is unique. This leaves the following question: How do we show that nested GKAT automata (automata that behave like GKAT expressions) admit solutions? Direction 2 is relevant here: It could be much easier to show that a smaller covariety of GKAT automata admit solutions instead. Nested GKAT automata are presented by the nesting coequation (Definition 3.1.31), so finding a smaller covariety amounts to refining the nesting coequation with additional colours.

**Direction 4: Completeness of GKAT without UA**

In Chapter 3, we proved that the axioms of GKAT are complete for the fragment consisting of propositional while programs that do not contain the skip command, i.e., skip-free GKAT. Without the uniqueness axiom, the completeness problem for full GKAT remains open, but our completeness results for skip-free GKAT are encouraging. The fact that we could reuse results by Grabmayer and Fokkink [GF20] has led us to consider the possibility of a full reduction from bisimulation GKAT (without the uniqueness axiom) to Grabmayer's completeness theorem for regular expressions modulo bisimilarity. Unfortunately, it is not clear what the reduction

[3] This is also the case for the completeness theorem for star fragments in Section 4.5.1 of Chapter 4, which also uses a uniqueness axiom. The key point is that the uniqueness axiom, a one-to-one correspondence between solutions and coalgebra homomorphisms, and the characterization of bisimilarity in terms of spans in Lemma 2.2.12 together imply that bisimilar systems admit equivalent solutions.

strategy should be: GKAT automata admit many exits (one for each atomic test), whereas regular expressions admit only one exit (successful termination). This is reflected at the level of the syntax, for example in the GKAT program $pb$ where $0 <_{\mathsf{BA}} b <_{\mathsf{BA}} 1$, and this causes difficulties when trying to define a translation from GKAT to regular expressions modulo bisimilarity. In other words, it is not clear how the Boolean tests should be encoded as regular expressions. This is a concrete barrier to naively defining a reduction like we were able to do in the skip-free case. In skip-free GKAT programs, exits occur either for all atomic tests or none, which can be encoded as $1$ and $0$ respectively.

This concrete barrier to naively translating the syntax of GKAT into regular expressions motivates us to ask whether a reduction is possible in the first place: Can the completeness of bisimulation GKAT be reduced to the completeness of Milner's axiomatization of regular expressions modulo bisimilarity? A step in the right direction might be to pull the class of well-layered precharts back along the natural inclusion of skip-free automata into the category of precharts and analyze the relationship between the well-layeredness conditions and the solvability of skip-free automata. I imagine this would reveal a connection between *well-layered skip-free automata* and the structures of equivalence proofs in GKAT, analogous to the coinductive formulation of Milner's axioms given by Grabmayer [Gra21].

**Direction 5: Bisimilarity with hypotheses**

In [KS96], Kozen establishes the completeness of KAT by observing that KAT is KA with additional axioms called *hypotheses* [Coh94]. In the case of KAT, the additional axioms express that a subset of the alphabet generates a Boolean algebra. I would be interested in seeing if an axiomatization of KAT programs modulo bisimilarity—call it *bisimulation* KAT—could be obtained by adding similar axioms to Milner's axiomatization of regular expressions modulo bisimilarity. One desirable outcome of such an approach would be a reduction of bisimulation GKAT to bisimulation KAT in the same spirit as the reduction of skip-free GKAT to one-free regular expressions.

I would generally be interested in seeing an example of hypotheses being used to axiomatize bisimilarity. Substantial effort has gone into developing general techniques for adding hypotheses to Kleene algebra and KAT [KM14; Dou+19; Kap+20; PRW21], axiomatizations of language equivalence, but no such theory has been

developed for axiomatizations of bisimilarity.

### 5.1.3 Effectful Process Algebras

In Chapter 4, I introduced a family of process types with branching structures determined by an algebraic theory. I provided each process type with a fully expressive specification language and paired it with a sound and complete axiomatization of behavioural equivalence. I furthermore exhibited bisimulation GKAT and regular expressions modulo bisimilarity as fragments of these expressive calculi, called *star fragments*. There are numerous directions for research regarding effectful process calculi and their star fragments.

**Direction 6: Structural operational semantics**

I would like to know whether our operational semantics for effectful process algebras is an instance of the mathematical operational semantics introduced by Turi and Plotkin [TP97]. More specifically, I would like to see monad-over-functor distributive laws $\Sigma_M^* B_M \Rightarrow B_M \Sigma_M^*$ and $\Omega_M^* L_M \Rightarrow L_M \Omega_M^*$ (where $\Omega$ is the star fragment algebraic signature, see Footnote 12) that correspond to our operational semantics. In the effectful process calculus case, it appears that involvement of variables poses a barrier to finding such a distributive law. One way around this might be the use of nominal sets (see for example [GP02]) in the representations of process terms and their behaviours, but more investigation is needed. On the surface, the situation for star expressions appears to be much simpler: It is easy to check, for example, that the Brzozowski-like derivative that gives GKAT its operational semantics is given by a distributive law, and similarly for regular expressions modulo bisimilarity (both are obtained from a GSOS specification [BIM95; TP97]).

**Direction 7: Capturing examples from the literature**

Two particular examples of effectful process calculi explored in Chapter 4 closely resemble process calculi from the literature. The effectful process calculus obtained from the theory of convex algebra (Example 4.2.8) has a syntax and operational semantics that closely resembles Stark and Smolka's probabilistic process calculus [SS00]. I conjecture that the two are indeed equivalent, and intend to verify this in future work. Similarly, the effectful process calculus obtained from the theory of convex semilattices with top (Example 4.1.15) closely resembles the calculus of

Mislove, Oaknine, and Worrell [MOW03]. Again, I conjecture they are equivalent.

**Direction 8: Completeness theorems for other fragments**

In Section 4.5, I introduce star fragments of expressive effectful process calculi. Star fragments offer a uniform construction of bisimulation variants of Kleene-like algebras for a variety of paradigms of computing. I leave as an open problem whether completeness theorems can be established for star fragments and their one-free fragments in Section 4.5.

Here, however, I would like to draw attention to the fact that star fragments are a somewhat arbitrary construction. For example, instead of star fragments, we could have considered the fragment of the effectful process calculus with a *perpetual loop operator* $(-)^\omega$, whose translation into the expressive calculus is defined

$$\mathrm{tr}(1) = \underline{u} \qquad \mathrm{tr}(c) = c \qquad \mathrm{tr}(e +_\sigma f) = \sigma(\mathrm{tr}(e), \mathrm{tr}(f))$$
$$\mathrm{tr}(ef) = \mathrm{tr}(e)[\mathrm{tr}(f)/\underline{u}] \qquad \mathrm{tr}(e^\omega) = \mu \underline{u}\ \mathrm{tr}(e)$$

A sound axiomatization of this fragment can be obtained from the properties of substitution and a restriction of Milner's axioms, as follows:

$$\frac{\mathsf{EQ} \vdash e = f}{e = f} \qquad 1e = e1 = e \quad ce = c \quad e(fg) = (ef)g \quad (e +_\sigma f)g = eg +_\sigma fg$$

$$e^\omega f = ee^\omega \qquad \frac{(\forall i \leq n)\ g_i \text{ is guarded}}{p(1,\vec{g})^\omega = (\mathsf{fp}\ \underline{u}\ p(\underline{u}, \vec{g}p(1,\vec{g})))^\omega} \qquad \frac{g = eg \qquad e \text{ is guarded}}{g = e^\omega}$$

Instantiated with semilattices, this is precisely Fokkink's *terminal cycle* fragment of regular expressions modulo bisimulation [Fok97]. In op. cit., Fokkink shows that an axiomatization equivalent to the one given above is complete for bisimilarity.

The purpose of exhibiting the perpetual loop fragment above is just to illustrate that fragments other than just the star fragment are of interest, and each fragment can be systematically axiomatized via a restriction of the axiomatization of the expressive calculus. I would specifically like to address *coalgebraic unital* fragments, which are formally captured as follows.

Let $(S, \mathsf{EQ}, \mathsf{fp})$ be a branching theory presenting $(M, \eta, \mu)$ and consider the $B_M$-

coalgebra $(Exp, \varepsilon)$ of effectful process expressions given by $(S, \mathsf{EQ}, \mathsf{fp})$. A *unital fixed-point structure* on $Exp$ is an algebraic signature $\{F_n\}_{n\in\mathbb{N}}$ paired with functions $\mathbf{rec}_n \colon F_n \times Exp^n \to Exp$ that do not increase the set of free variables. That is, if $V$ is the set of free variables that appear in $e_1, \ldots, e_n$, then $V$ contains the set of free variables in $\mathbf{rec}_n(\tau, e_1, \ldots, e_n)$. The functions $\mathbf{rec}_n$ should be thought of as assigning to each $\tau \in F_n$ a format for a recursive operation (like the Kleene star, or a while loop).

Fix two distinguished variables $\underline{u}, v \in Var$ called the *unit* and *dummy variable*. The *unital fragment* of $(Exp, \varepsilon)$ corresponding to a unital fixed-point structure is the set of expressions $Exp_{\mathbf{rec}}$ given by

$$Exp_{\mathbf{rec}} \ni e_1, e_2, e_i ::= \underline{u} \mid \sigma(e_1, \ldots, e_n) \mid e_1 e_2 \mid \tau(e_1, \ldots, e_n)$$

where $\sigma \in S_n$ and $\tau \in F_n$. The fragment-to-process term interpretation $ftp$ is given by

$$\begin{aligned} ftp(\underline{u}) &= \underline{u} \\ ftp(\sigma(e_1, \ldots, e_n)) &= \sigma(ftp(e_1), \ldots, ftp(e_n)) \\ ftp(e_1 e_2) &= ftp(e_1)[ftp(e_2)/\underline{u}] \\ ftp(\tau(e_1, \ldots, e_n)) &= \mu v\ \mathbf{rec}_n(\tau, ftp(e_1)[v/\underline{u}], \ldots, ftp(e_n)[v/\underline{u}]) \end{aligned}$$

We say that $Exp_{\mathbf{rec}}$ is a *coalgebraic unital fragment* if it is a subcoalgebra of $(Exp, \varepsilon)$.

For example, star fragments are obtained by setting $F_1 = \{(-)^{(\sigma)} \mid \sigma \in S_2\}$ and $\mathbf{rec}_1((-)^{(\sigma)}, e_1) = \sigma(e_1, \underline{u})$. The full calculus is given by setting $F_n$ to be the set of all process expressions with $n$ free variables (in which neither $\underline{u}$ nor $v$ appear bound) and $\mathbf{rec}_n$ to be substitution. The base calculus, appearing at the bottom of Figure 5.1, is obtained by setting $F_n = \emptyset$ for all $n \in \mathbb{N}$.

Coalgebraic unital fragments can be arranged in a diagram like Figure 5.1, with the expressive calculus at the top and the base fragment ($F_n = \emptyset$ for all $n$) at the bottom. In Figure 5.1, fragments are ordered by *behavioural expressiveness*: A fragment $Exp_1$ is below another fragment $Exp_2$ if we have the inclusion $!_\varepsilon(Exp_1) \subseteq !_\varepsilon(Exp_2)$, where $!_\varepsilon$ is the final coalgebra map $(Exp, \varepsilon) \to (Z, \zeta)$. I have only written the names of five calculi in that diagram, but there are in principle many others—in-between the ones I have explicitly written, or incomparable.

The main question I have about these fragments is their behavioural expres-

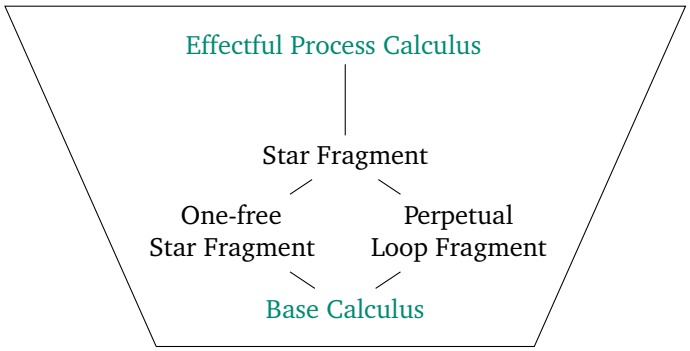



**Figure 5.1:** Fragments of the expressive effectful process calculus.

siveness: Which behavioural coequations are the sets of behaviours specified by coalgebraic unital fragments? In particular, I would like to see a behavioural covariety that is *not* the set of behaviours of a coalgebraic unital fragment.

#### Direction 9: Higher arity actions

One way to interpret action symbols in our process calculi is as basic program tasks. Single arity action symbols are useful for modelling processes where basic program tasks are executed sequentially, one after the other. To model program tasks that are more complex, perhaps due to splitting into subroutines, we need to allow action symbols to have multiple arguments. Consider, for example, the process $\mathsf{start\_baking}(\mathsf{add\_ingredients}\ v_1, \mathsf{mix}\ v_2, \mathsf{bake}\ v_3)$. The action symbol $\mathsf{start\_baking}$ represents a task with three steps, the order of which matters modulo bisimilarity. The (output) variable $v_i$ should be seen as an indication that "step $i$" has been completed.

If we replace $Act \times \mathrm{Id}$ in our signatures $\Sigma_M$, $B_M$, and $L_M$ with an arbitrary polynomial functor, allowing action symbols to have higher arity than one, do we obtain semantics, axiomatizations, and completeness theorems like in the effectful process algebra case? I conjecture that the answer is yes, at least for effectful process calculi. There are essentially two reasons behind this conjecture: Firstly, the relevant notion of guardedness is easily reformulated to fit the multiple arity case, and so is the axiomatization of bisimilarity in Section 4.4. Secondly, and more importantly, the completeness proof for effectful process calculi (essentially a generalization of Milner's completeness proof for the algebra of regular behaviours [Mil84]) does not depend on the singular arity of action symbols.

**Direction 10: Axiomatizing Other Equivalences and Relations**

For the most part, the thesis focuses on axiomatizations of bisimilarity. Bisimilarity is only one of many relations between programs one might want to reason about. For example, van Glabbeek's linear time–branching time spectrum from process algebra illustrates an intricate lattice of equivalences that lie in-between trace and bisimilarity, ordered by inclusion [Gla90; Gla93] (some of which have been formulated coalgebraically [Mon08; Cal14] and with the use of graded monads [MPS15; DMS19; FMS21; For+22]). A different kind of relation one might want to reason about is *similarity* [HJ04; Lev11], a sort of one-sided bisimilarity that in the regular language case corresponds to inclusion. In certain cases, similarity can be axiomatized by considering coalgebras in the category of partially ordered sets and monotone maps (see [Sch22a] for the specific case where $\mathsf{fp}\, v\ p(v,\vec{x})$ is computed as a least fixed-point). A third kind of relation is behavioural distance, common in probabilistic computation [Des99; BW05; BGP17]. I conjecture that an approach similar to the one in this thesis can be taken to axiomatizing behavioural distances for program branching characterized by a quantitative equational theory [MPP21]. Furthermore, I expect that the axiomatization problems for similarity and behavioural distance can be unified in the context of quantale-enriched algebraic theories [Pow99].

**Direction 11: Impossibility Results and Other Kinds of Axiomatizations**

Redko proved in [Red64] that there is no finite equational axiomatization of language equivalence for regular expressions. Similar results exist in process algebra: For example, Aceto, Fokkink, Ingólfsdóttir, and Luttik proved in [Ace+05] that CCS with merge has no finite equational axiomatization, thereby confirming a conjecture made by Bergstra and Klop [BK84]. In light of these negative results, I conjecture that finite axiomatizations of bisimilarity only exist for trivial effectful process algebras, like the Base Calculus in Figure 5.1.

Using the syntax of string diagrams instead of regular expressions, Piedeleu and Zanasi give a finite axiomatization of language equivalence for finite state automata in [PZ23] [4]. A similar line of work treats Milner's merge operation from CCS [Mil80] (as well as Hoare's merge operation [Hoa78]) string-diagrammatically [Bon+19]. I

[4] Intuitively, the source of Redko's negative result is the Kleene star, which is a necessary ingredient in Kleene's theorem. For contrast, the Kleene star is a derived concept in Piedeleu and Zanasi's calculus, whose syntax is much closer to the automata-theoretic semantics of regular expressions.

conjecture that a diagrammatic approach to effectful process algebra exists, given that string diagrams have been used as a graphical syntax for traced monoidal categories [JSV96], which are categorical models of recursion (see for example [Has97]) that are deeply connected to iteration theories [SP00; BH03] (see the text under *Iteration Theories and Elgot Monads* in Section 4.6).

# Bibliography


[Ace+05] Luca Aceto, Wan J. Fokkink, Anna Ingólfsdóttir, and Bas Luttik. "CCS with Hennessy's merge has no finite-equational axiomatization". In: *Theor. Comput. Sci.* 330.3 (2005), pp. 377–405 (cit. on p. 250).

[Ace+11a] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. "Axiomatizing GSOS with Predicates". In: *SOS*. 2011, pp. 1–15 (cit. on p. 166).

[Ace+11b] Luca Aceto, Georgiana Caltais, Eugen-Ioan Goriac, and Anna Ingólfsdóttir. "PREG Axiomatizer - A Ground Bisimilarity Checker for GSOS with Predicates". In: *CALCO*. 2011, pp. 378–385 (cit. on p. 166).

[Ace94] Luca Aceto. "Deriving Complete Inference Systems for a Class of GSOS Languages Generation Regular Behaviours". In: *CONCUR*. 1994, pp. 449–464 (cit. on p. 166).

[Acz+03] Peter Aczel, Jirí Adámek, Stefan Milius, and Jiri Velebil. "Infinite trees and completely iterative theories: a coalgebraic view". In: *Theor. Comput. Sci.* 300.1-3 (2003), pp. 1–45 (cit. on p. 196).

[Adá03] Jiří Adámek. "On final coalgebras of continuous functors". In: *Theoretical Computer Science* 294.1 (2003). Category Theory and Computer Science, pp. 3–29. ISSN: 0304-3975 (cit. on p. 198).

[Adá05] Jiří Adámek. "Introduction to Coalgebra". In: *Theory and Applications of Categories [electronic only]* 14 (2005), pp. 157–199 (cit. on pp. 27, 45, 47, 188).

[AMV11] Jiří Adámek, Stefan Milius, and Jiří Velebil. "Elgot theories: a new perspective on the equational properties of iteration". In: *Math. Struct. Comput. Sci.* 21.2 (2011), pp. 417–480 (cit. on pp. 27, 239).

[AN80] André Arnold and Maurice Nivat. “The metric space of infinite trees. Algebraic and topological properties”. In: *Fundam. Informaticae* 3 (1980), pp. 445–476 (cit. on p. 198).

[And+14] Carolyn Jane Anderson, Nate Foster, Arjun Guha, Jean-Baptiste Jeannin, Dexter Kozen, Cole Schlesinger, and David Walker. “NetKAT: semantic foundations for networks”. In: *POPL*. 2014, pp. 113–126 (cit. on pp. 8, 20, 75, 167, 168).

[And99] Suzana Andova. “Process Algebra with Probabilistic Choice”. In: *ARTS*. Vol. 1601. Lecture Notes in Computer Science. Springer, 1999, pp. 111–129 (cit. on pp. 8, 169, 171).

[Ant96] Valentin M. Antimirov. “Partial Derivatives of Regular Expressions and Finite Automaton Constructions”. In: *Theoretical Computer Science* 155.2 (1996), pp. 291–319 (cit. on pp. 38, 59).

[AR94] Jiří Adámek and Jiří Rosický. *Locally Presentable and Accessible Categories*. London Mathematical Society Lecture Note Series. Cambridge University Press, 1994 (cit. on pp. 67, 188).

[Bae05] Jos C. M. Baeten. “A brief history of process algebra”. In: *Theoretical Computer Science* 335.2-3 (2005), pp. 131–146 (cit. on p. 169).

[Ban22] Stefan Banach. “Sur les opérations dans les ensembles abstraits et leur application aux équations intégrales”. In: *Fundamenta Mathematicae* 3 (1922), pp. 133–181 (cit. on p. 200).

[Bar93] Michael Barr. “Terminal Coalgebras in Well-Founded Set Theory”. In: *Theoretical Computer Science* 114.2 (1993), pp. 299–315 (cit. on pp. 189, 198).

[BBK87] Jos C. M. Baeten, Jan A. Bergstra, and Jan Willem Klop. “On the Consistency of Koomen’s Fair Abstraction Rule”. In: *Theoretical Computer Science* 51 (1987), pp. 129–176 (cit. on pp. 76, 224).

[BBR09] Jos C. M. Baeten, Twan Basten, and Michel A. Reniers. *Process Algebra: Equational Theories of Communicating Processes*. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2009 (cit. on p. 169).

[BCG06] Jos C. M. Baeten, Flavio Corradini, and Clemens Grabmayer. “On the Star Height of Regular Expressions Under Bisimulation (Extended Abstract)”. In: *EXPRESS*. Bonn, Germany, 2006 (cit. on pp. 22, 171, 239).

[BCG07] Jos C. M. Baeten, Flavio Corradini, and Clemens Grabmayer. “A characterization of regular expressions under bisimulation”. In: *J. ACM* 54.2 (2007), p. 6 (cit. on pp. 22, 27, 31, 40, 240).

[BE76] Stephen L. Bloom and Calvin C. Elgot. “The Existence and Construction of Free Iterative Theories”. In: *J. Comput. Syst. Sci.* 12.3 (1976), pp. 305–318 (cit. on pp. 27, 230, 239).

[BÉ88] Stephen L. Bloom and Zoltán Ésik. “Varieties of Iteration Theories”. In: *SIAM J. Comput.* 17.5 (1988), pp. 939–966 (cit. on pp. 27, 179, 239).

[BÉ93] Stephen L. Bloom and Zoltán Ésik. *Iteration Theories - The Equational Logic of Iterative Processes*. EATCS Monographs on Theoretical Computer Science. Springer, 1993 (cit. on p. 165).

[Bec69] Jon Beck. “Distributive laws”. In: *Seminar on triples and categorical homology theory*. Springer. 1969, pp. 119–140 (cit. on p. 175).

[Bek84] Hans Bekić. “Definable operations in general algebras, and the theory of automata and flowcharts”. In: *Programming Languages and Their Definition: H. Bekič (1936–1982)*. Ed. by C. B. Jones. Berlin, Heidelberg: Springer Berlin Heidelberg, 1984, pp. 30–55 (cit. on p. 240).

[Ben77] Johann van Benthem. “Modal Correspondence Theory”. PhD thesis. University of Amsterdam, 1977 (cit. on p. 22).

[BGP17] Borja Balle, Pascale Gourdeau, and Prakash Panangaden. “Bisimulation Metrics for Weighted Automata”. In: *ICALP*. Vol. 80. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 103:1–103:14 (cit. on p. 250).

[BH03] Nick Benton and Martin Hyland. “Traced Premonoidal Categories”. In: *RAIRO - Theoretical Informatics and Applications* 37.4 (2003), pp. 273–299 (cit. on p. 251).

[BIM95] Bard Bloom, Sorin Istrail, and Albert R. Meyer. “Bisimulation Can’t be Traced”. In: *J. ACM* 42.1 (1995), pp. 232–268 (cit. on p. 246).

[Bir35] Garrett Birkhoff. “On the Structure of Abstract Algebras”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 31.4 (1935), pp. 433–454 (cit. on pp. 175, 243).

[BK82] Jan A. Bergstra and Jan Willem Klop. “Fixed point semantics in process algebras (preprint)”. Jan. 1982 (cit. on p. 22).

[BK84] J.A. Bergstra and J.W. Klop. “Process algebra for synchronous communication”. In: *Information and Control* 60.1 (1984), pp. 109–137. ISSN: 0019-9958 (cit. on p. 250).

[BK88] Jan A. Bergstra and Jan Willem Klop. “Process theory based on bisimulation semantics”. In: *REX Workshop*. Vol. 354. Lecture Notes in Computer Science. Springer, 1988, pp. 50–122 (cit. on p. 171).

[BM96] Jon Barwise and Lawrence S. Moss. *Vicious circles - on the mathematics of non-wellfounded phenomena*. Vol. 60. CSLI lecture notes series. CSLI, 1996 (cit. on p. 230).

[BMS13] Marcello M. Bonsangue, Stefan Milius, and Alexandra Silva. “Sound and Complete Axiomatizations of Coalgebraic Language Equivalence”. In: *ACM Trans. Comput. Logic* 14.1 (Feb. 2013) (cit. on pp. 27, 33, 34, 66, 67, 72, 239).

[Bon+12] Filippo Bonchi, Marcello Bonsangue, Michele Boreale, Jan Rutten, and Alexandra Silva. “A coalgebraic perspective on linear weighted automata”. In: *Information and Computation* 211 (2012), pp. 77–105 (cit. on p. 45).

[Bon+19] Filippo Bonchi, Robin Piedeleu, Pawel Sobocinski, and Fabio Zanasi. “Bialgebraic Semantics for String Diagrams”. In: *CONCUR*. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 37:1–37:17 (cit. on p. 250).

[BP13] Filippo Bonchi and Damien Pous. “Checking NFA equivalence with bisimulations up to congruence”. In: *POPL*. ACM, 2013, pp. 457–468 (cit. on p. 22).

[Brz64] Janusz A. Brzozowski. “Derivatives of Regular Expressions”. In: *J. ACM* 11.4 (1964), pp. 481–494 (cit. on pp. 52, 87, 88, 114, 184).

[BSS17] Filippo Bonchi, Alexandra Silva, and Ana Sokolova. “The Power of Convex Algebras”. In: *CONCUR*. Vol. 85. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2017, 23:1–23:18 (cit. on p. 181).

[BSV19] Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. “The Theory of Traces for Systems with Nondeterminism and Probability”. In: *LICS*. IEEE, 2019, pp. 1–14 (cit. on pp. 169, 175, 183).

[BSV21] Filippo Bonchi, Ana Sokolova, and Valeria Vignudelli. “Presenting Convex Sets of Probability Distributions by Convex Semilattices and Unique Bases ((Co)algebraic pearls)”. In: *CALCO*. Vol. 211. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 11:1–11:18 (cit. on pp. 169, 175, 179, 181).

[BT83] Stephen L. Bloom and Ralph Tindell. “Varieties of ‘if-then-else’”. In: *SIAM J. Comput.* 12.4 (1983), pp. 677–707 (cit. on pp. 170, 179).

[BW05] Franck van Breugel and James Worrell. “A behavioural pseudometric for probabilistic transition systems”. In: *Theor. Comput. Sci.* 331.1 (2005), pp. 115–142 (cit. on p. 250).

[BW81] Manfred Broy and Martin Wirsing. “On the Algebraic Specification of Nondeterministic Programming Languages”. In: *CAAP*. Vol. 112. Lecture Notes in Computer Science. Springer, 1981, pp. 162–179 (cit. on p. 173).

[BW90] Jos C. M. Baeten and W. P. Weijland. *Process algebra*. Vol. 18. Cambridge tracts in theoretical computer science. Cambridge University Press, 1990 (cit. on pp. 22, 165, 169).

[Cal14] Georgiana Caltais. “Coalgebraic Tools for Bisimilarity and Decorated Trace Semantics”. Reykjavík University, 2014 (cit. on p. 250).

[Cha+19] Tej Chajed, Joseph Tassarotti, M. Frans Kaashoek, and Nickolai Zeldovich. “Argosy: verifying layered storage systems with recovery refinement”. In: *PLDI*. 2019, pp. 1054–1068 (cit. on p. 75).

[Cîr17] Corina Cîrstea. “From Branching to Linear Time, Coalgebraically”. In: *Fundam. Informaticae* 150.3-4 (2017), pp. 379–406 (cit. on pp. 169, 238).

[CKS96] Ernie Cohen, Dexter Kozen, and Frederick Smith. *The complexity of Kleene algebra with tests*. Tech. rep. Cornell University, Ithaca, USA, 1996 (cit. on p. 21).

[Coh09] Ernie Cohen. *Weak Kleene Algebra is Sound and (Possibly) Complete for Simulation*. 2009 (cit. on p. 166).

[Coh94] Ernie Cohen. *Hypotheses in Kleene Algebra*. Tech. rep. Bellcore, 1994 (cit. on pp. 167, 245).

[Con71] John Horton Conway. *Regular Algebra and Finite Machines*. Chapman and Hall, London, UK, 1971 (cit. on pp. 20, 34, 66).

[Cou83] Bruno Courcelle. "Fundamental Properties of Infinite Trees". In: *Theor. Comput. Sci.* 25 (1983), pp. 95–169 (cit. on p. 196).

[Des99] Josée Desharnais. "Logical characterization of simulation for labelled Markov chains". In: *Proceedings of PROBMIV'99* (1999) (cit. on p. 250).

[DMS19] Ulrich Dorsch, Stefan Milius, and Lutz Schröder. "Graded Monads and Graded Logics for the Linear Time - Branching Time Spectrum". In: *CONCUR*. Vol. 140. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019, 36:1–36:16 (cit. on p. 250).

[Dou+19] Amina Doumane, Denis Kuperberg, Damien Pous, and Pierre Pradic. "Kleene Algebra with Hypotheses". In: *FOSSACS*. 2019, pp. 207–223 (cit. on pp. 167, 245).

[DS21] Fredrik Dahlqvist and Todd Schmid. "How to Write a Coequation ((Co)algebraic pearls)". In: *CALCO*. Vol. 211. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 13:1–13:25 (cit. on p. 243).

[Elg75] Calvin C. Elgot. "Monadic Computation And Iterative Algebraic Theories". In: *Logic Colloquium '73*. Ed. by Harvey E. Rose and John C. Shepherdson. Vol. 80. Studies in Logic and the Foundations of Mathematics. Elsevier, 1975, pp. 175–230 (cit. on pp. 27, 239).

[FMS21] Chase Ford, Stefan Milius, and Lutz Schröder. "Behavioural Preorders via Graded Monads". In: *36th Annual ACM/IEEE Symposium on Logic in Computer Science, LICS 2021, Rome, Italy, June 29 - July 2, 2021*. IEEE, 2021, pp. 1–13 (cit. on p. 250).

[Fok97] Wan J. Fokkink. “Axiomatizations for the Perpetual Loop in Process Algebra”. In: *ICALP*. Vol. 1256. Lecture Notes in Computer Science. Springer, 1997, pp. 571–581 (cit. on pp. 22, 171, 239, 247).

[For+22] Chase Ford, Stefan Milius, Lutz Schröder, Harsh Beohar, and Barbara König. “Graded Monads and Behavioural Equivalence Games”. In: *LICS*. ACM, 2022, 61:1–61:13 (cit. on p. 250).

[Fos+15] Nate Foster, Dexter Kozen, Mae Milano, Alexandra Silva, and Laure Thompson. “A Coalgebraic Decision Procedure for NetKAT”. In: *POPL*. ACM, 2015, pp. 343–355 (cit. on pp. 8, 75, 165).

[Fos+16] Nate Foster, Dexter Kozen, Konstantinos Mamouras, Mark Reitblatt, and Alexandra Silva. “Probabilistic NetKAT”. In: *ESOP*. Vol. 9632. Lecture Notes in Computer Science. Springer, 2016, pp. 282–309 (cit. on pp. 8, 75, 168).

[FS15] Simon Forster and Georg Struth. “On the Fine-Structure of Regular Algebra”. In: *Journal of Automatic Reasoning* 54 (2015), pp. 165–197 (cit. on p. 20).

[FZ94] Wan J. Fokkink and Hans Zantema. “Basic Process Algebra with Iteration: Completeness of its Equational Axioms”. In: *Comput. J.* 37.4 (1994), pp. 259–268 (cit. on pp. 22, 171, 239).

[FZ97] Wan J. Fokkink and Hans Zantema. “Termination Modulo Equations by Abstract Commutation with an Application to Iteration”. In: *Theoretical Computer Science* 177.2 (1997), pp. 407–423 (cit. on pp. 171, 239).

[GF20] Clemens Grabmayer and Wan J. Fokkink. “A Complete Proof System for 1-Free Regular Expressions Modulo Bisimilarity”. In: *LICS*. ACM, 2020, pp. 465–478 (cit. on pp. 22, 28, 31–34, 39, 42, 49, 50, 53, 55, 56, 58, 59, 61–64, 66, 72, 77, 78, 135, 166, 167, 171, 239–241, 244).

[Gla90] Rob J. van Glabbeek. “The Linear Time-Branching Time Spectrum (Extended Abstract)”. In: *CONCUR*. Vol. 458. Lecture Notes in Computer Science. Springer, 1990, pp. 278–297 (cit. on p. 250).

[Gla93] Rob J. van Glabbeek. “The Linear Time - Branching Time Spectrum II”. In: *CONCUR ’93, 4th International Conference on Concurrency Theory, Hildesheim, Germany, August 23-26, 1993, Proceedings*. Vol. 715. Lecture Notes in Computer Science. Springer, 1993, pp. 66–81 (cit. on p. 250).

[Gog+77] Joseph A. Goguen, James W. Thatcher, Eric G. Wagner, and Jesse B. Wright. “Initial Algebra Semantics and Continuous Algebras”. In: *J. ACM* 24.1 (1977), pp. 68–95 (cit. on p. 191).

[GP02] Murdoch Gabbay and Andrew M. Pitts. “A New Approach to Abstract Syntax with Variable Binding”. In: *Formal Aspects Comput.* 13.3-5 (2002), pp. 341–363 (cit. on p. 246).

[Gra05] Clemens Grabmayer. “Using Proofs by Coinduction to Find ‘Traditional’ Proofs”. In: *CALCO*. Vol. 3629. Lecture Notes in Computer Science. Springer, 2005, pp. 175–193 (cit. on p. 22).

[Gra21] Clemens Grabmayer. “A Coinductive Version of Milner’s Proof System for Regular Expressions Modulo Bisimilarity”. In: *CALCO*. Vol. 211. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 16:1–16:23 (cit. on pp. 22, 39, 171, 245).

[Gra22] Clemens Armin Grabmayer. “Milner’s Proof System for Regular Expressions Modulo Bisimilarity is Complete: Crystallization: Near-Collapsing Process Graph Interpretations of Regular Expressions”. In: *LICS*. ACM, 2022, 34:1–34:13 (cit. on pp. 22, 23, 26, 31, 33, 39, 166–168, 171, 230, 239, 240, 242).

[GS01] H. Peter Gumm and Tobias Schröder. “Covarieties and complete covarieties”. In: *Theoretical Computer Science* 260.1-2 (2001), pp. 71–86 (cit. on p. 243).

[Gum98] H. Peter Gumm. “Functors for Coalgebras”. In: *Algebra Universalis* 45 (Nov. 1998) (cit. on pp. 48, 89).

[Gum99] H. Peter Gumm. “Elements of the general theory of coalgebras”. In: *LUATCS’99, Rand Afrikaans University, Johannesburg* (1999) (cit. on pp. 27, 45, 47, 69, 218).

[Has97] Masahito Hasegawa. “Models of sharing graphs: a categorical semantics of let and letrec”. PhD thesis. University of Edinburgh, UK, 1997 (cit. on p. 251).

[HJ04] Jesse Hughes and Bart Jacobs. “Simulations in coalgebra”. In: *Theoretical Computer Science* 327.1-2 (2004), pp. 71–108 (cit. on p. 250).

[HK73] John E. Hopcroft and Richard M. Karp. “An $n^{5/2}$ Algorithm for Maximum Matchings in Bipartite Graphs”. In: *SIAM J. Comput.* 2.4 (1973), pp. 225–231 (cit. on p. 21).

[HM12] Peter Höfner and Bernhard Möller. “Dijkstra, Floyd and Warshall meet Kleene”. In: *Formal Aspects Comput.* 24.4-6 (2012), pp. 459–476 (cit. on p. 20).

[Hoa+87] Charles A. R. Hoare, Ian J. Hayes, Jifeng He, Carroll Morgan, A. W. Roscoe, Jeff W. Sanders, Ib Holm Sorensen, J. Michael Spivey, and Bernard Sufrin. “Laws of Programming”. In: *Commun. ACM* 30.8 (1987), pp. 672–686 (cit. on p. 19).

[Hoa78] Charles A. R. Hoare. “Communicating Sequential Processes”. In: *Commun. ACM* 21.8 (1978), pp. 666–677 (cit. on p. 250).

[Hoa80] Charles A. R. Hoare. “A Model for Communicating Sequential Processes”. In: *On the Construction of Programs*. Ed. by R. M. McKeag and A. M. Macnaghten. Cambridge University Press, 1980, pp. 229–254 (cit. on p. 22).

[HS96] Hans Huttel and Shukla Scott. *On the Complexity of Deciding Behavioural Equivalences and Preorders*. Tech. rep. USA, 1996 (cit. on p. 22).

[Hun04] Edward V. Huntington. “Sets of independent postulates for the algebra of logic”. In: *Transactions of the American Mathematical Society* 5.3 (1904), pp. 288–309 (cit. on p. 93).

[Jac06] Bart Jacobs. “A Bialgebraic Review of Deterministic Automata, Regular Expressions and Languages”. In: *Essays Dedicated to Joseph A. Goguen*. Vol. 4060. Lecture Notes in Computer Science. Springer, 2006, pp. 375–404 (cit. on pp. 28, 32–34, 66, 67, 72).

[Jac10] Bart Jacobs. “Convexity, Duality and Effects”. In: *IFIP TCS*. Vol. 323. IFIP Advances in Information and Communication Technology. Springer, 2010, pp. 1–19 (cit. on p. 169).

[Jac16] Bart Jacobs. *Introduction to Coalgebra: Towards Mathematics of States and Observation*. Vol. 59. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, 2016 (cit. on pp. 23, 27, 33, 45, 47).

[Joh82] Peter T. Johnstone. *Stone spaces*. Cambridge Studies in Advanced Mathematics 3. Cambridge: Cambridge University Press, 1982 (cit. on p. 175).

[JSV96] André Joyal, Ross Street, and Dominic Verity. “Traced monoidal categories”. In: *Mathematical Proceedings of the Cambridge Philosophical Society* 119.3 (1996), pp. 447–468 (cit. on p. 251).

[Kap+18] Tobias Kappé, Paul Brunet, Alexandra Silva, and Fabio Zanasi. “Concurrent Kleene Algebra: Free Model and Completeness”. In: *ESOP*. Vol. 10801. Lecture Notes in Computer Science. Springer, 2018, pp. 856–882 (cit. on pp. 20, 167).

[Kap+19] Tobias Kappé, Paul Brunet, Jurriaan Rot, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. “Kleene Algebra with Observations”. In: *CONCUR*. 2019, 41:1–41:16 (cit. on pp. 165, 167).

[Kap+20] Tobias Kappé, Paul Brunet, Alexandra Silva, Jana Wagemaker, and Fabio Zanasi. “Concurrent Kleene Algebra with Observations: From Hypotheses to Completeness”. In: *FOSSACS*. 2020, pp. 381–400 (cit. on pp. 165, 167, 245).

[KK05] Lucja Kot and Dexter Kozen. “Kleene Algebra and Bytecode Verification”. In: *Electron. Notes Theoretical Computer Science* 141.1 (2005), pp. 221–236 (cit. on p. 75).

[Kle56] Stephen Cole Kleene. “Representation of Events in Nerve Nets and Finite Automata”. In: *Automata Studies*. Princeton University Press, 1956 (cit. on pp. 20, 31, 34, 35, 42, 55, 75, 113, 165).

[KM14] Dexter Kozen and Konstantinos Mamouras. “Kleene Algebra with Equations”. In: *ICALP*. 2014, pp. 280–292 (cit. on pp. 167, 245).

[Koc70] Anders Kock. “Monads on symmetric monoidal closed categories”. In: *Archiv der Mathematik* 21 (1 1970), pp. 1–10 (cit. on p. 238).

[Koz91] Dexter Kozen. “A Completeness Theorem for Kleene Algebras and the Algebra of Regular Events”. In: *LICS*. IEEE Computer Society, 1991, pp. 214–225 (cit. on pp. 20, 33, 55, 66, 76, 165).

[Koz96] Dexter Kozen. “Kleene Algebra with Tests and Commutativity Conditions”. In: *TACAS*. 1996, pp. 14–33 (cit. on pp. 20, 21).

[KP00] Dexter Kozen and Maria-Christina Patron. “Certification of Compiler Optimizations Using Kleene Algebra with Tests”. In: *CL*. 2000, pp. 568–582 (cit. on pp. 20, 75).

[Kro90] Daniel Krob. “A Complete System of B-Rational Identities”. In: *ICALP*. Vol. 443. Lecture Notes in Computer Science. Springer, 1990, pp. 60–73 (cit. on p. 20).

[KS96] Dexter Kozen and Frederick Smith. “Kleene Algebra with Tests: Completeness and Decidability”. In: *CSL*. Vol. 1258. Lecture Notes in Computer Science. Springer, 1996, pp. 244–259 (cit. on pp. 20, 21, 75, 165, 166, 245).

[KSS23] Tobias Kappé, Todd Schmid, and Alexandra Silva. “A Complete Inference System for Skip-free Guarded Kleene Algebra with Tests”. In: *ESOP*. Vol. 13990. Lecture Notes in Computer Science. Springer, 2023, pp. 309–336 (cit. on pp. 27, 29, 76, 78, 225).

[KT08] Dexter Kozen and Wei-Lung Dustin Tseng. “The Böhm-Jacopini Theorem Is False, Propositionally”. In: *MPC*. Vol. 5133. Lecture Notes in Computer Science. Springer, 2008, pp. 177–192 (cit. on pp. 19, 21, 76, 84, 90, 165, 225).

[Lam68] Joachim Lambek. “A fixpoint theorem for complete categories”. In: *Mathematische Zeitschrift* 103.2 (1968), pp. 151–161 (cit. on pp. 67, 190, 194).

[Law63] F William Lawvere. “Functorial semantics of algebraic theories”. In: *Proceedings of the National Academy of Sciences of the United States of America* 50.5 (1963), p. 869 (cit. on p. 169).

[Lev11] Paul Blain Levy. “Similarity Quotients as Final Coalgebras”. In: *FoSSaCS*. Vol. 6604. Lecture Notes in Computer Science. Springer, 2011, pp. 27–41 (cit. on p. 250).

[LS17] Michael R Laurence and Georg Struth. *Completeness Theorems for Pomset Languages and Concurrent Kleene Algebras*. 2017. arXiv: 1705.05896 [cs.FL] (cit. on p. 167).

[Mak87] Johann A. Makowsky. “Why Horn Formulas Matter in Computer Science: Initial Structures and Generic Examples”. In: *J. Comput. Syst. Sci.* 34.2/3 (1987), pp. 266–292 (cit. on p. 165).

[Man76] Ernest G. Manes. *Algebraic Theories*. 1st ed. Graduate Texts in Mathematics. Springer-Verlag New York, 1976 (cit. on pp. 169, 175).

[Man91] Ernest G. Manes. “Equations for if-then-else”. In: *MFPS*. Vol. 598. Lecture Notes in Computer Science. Springer, 1991, pp. 446–456 (cit. on pp. 25, 170, 179).

[McC61] John McCarthy. “A basis for a mathematical theory of computation, preliminary report”. In: *IRE-AIEE-ACM Computer Conference (Western)*. ACM, 1961, pp. 225–238 (cit. on pp. 170, 179).

[Mil10] Stefan Milius. “A Sound and Complete Calculus for Finite Stream Circuits”. In: *LICS*. IEEE Computer Society, 2010, pp. 421–430 (cit. on pp. 27, 33, 34, 45, 66, 67, 239).

[Mil80] Robin Milner. *A Calculus of Communicating Systems*. Vol. 92. Lecture Notes in Computer Science. Springer, 1980 (cit. on pp. 22, 37, 169, 250).

[Mil84] Robin Milner. “A Complete Inference System for a Class of Regular Behaviours”. In: *J. Comput. Syst. Sci.* 28.3 (1984), pp. 439–466 (cit. on pp. 8, 19, 22, 24, 26, 27, 31, 39, 40, 42, 49, 165, 166, 169–171, 173, 176, 184, 221, 237, 238, 240, 242, 249).

[MN87] Alan H. Mekler and Evelyn M. Nelson. “Equational Bases for If-Then-Else”. In: *SIAM Journal on Computing* 16.3 (1987), pp. 465–485 (cit. on p. 179).

[Mog89] Eugenio Moggi. “Computational Lambda-Calculus and Monads”. In: *Proceedings of the Fourth Annual Symposium on Logic in Computer Science (LICS ’89), Pacific Grove, California, USA, June 5-8, 1989*. IEEE Computer Society, 1989, pp. 14–23 (cit. on p. 237).

[Mog91] Eugenio Moggi. “Notions of computation and monads”. In: *Information and Computation* 93.1 (1991). Selections from 1989 IEEE Symposium on Logic in Computer Science, pp. 55–92. ISSN: 0890-5401 (cit. on pp. 25, 237).

[Mon08] Luís Monteiro. “A Coalgebraic Characterization of Behaviours in the Linear Time - Branching Time Spectrum”. In: *WADT*. Vol. 5486. Lecture Notes in Computer Science. Springer, 2008, pp. 251–265 (cit. on p. 250).

[MOW03] Michael W. Mislove, Joël Ouaknine, and James Worrell. “Axioms for Probability and Nondeterminism”. In: *EXPRESS*. Vol. 96. Electronic Notes in Theoretical Computer Science. Elsevier, 2003, pp. 7–28 (cit. on pp. 8, 183, 237, 247).

[MPP21] Radu Mardare, Prakash Panangaden, and Gordon Plotkin. “Fixed-Points for Quantitative Equational Logics”. In: *2021 36th Annual ACM/IEEE Symposium on Logic in Computer Science (LICS)* (June 2021) (cit. on p. 250).

[MPS15] Stefan Milius, Dirk Pattinson, and Lutz Schröder. “Generic Trace Semantics and Graded Monads”. In: *CALCO*. Vol. 35. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2015, pp. 253–269 (cit. on p. 250).

[Mye09] Robert S. R. Myers. “Coalgebraic Expressions”. In: *FICS*. Institute of Cybernetics, 2009, pp. 61–69 (cit. on p. 239).

[Nel83] Evelyn Nelson. “Iterative Algebras”. In: *Theoretical Computer Science* 25 (1983), pp. 67–94 (cit. on pp. 27, 239, 240).

[Par81] David Michael Ritchie Park. “Concurrency and Automata on Infinite Sequences”. In: *Theoretical Computer Science*. Vol. 104. Lecture Notes in Computer Science. Springer, 1981, pp. 167–183 (cit. on p. 22).

[Plo04] Gordon D. Plotkin. “A structural approach to operational semantics”. In: *J. Log. Algebraic Methods Program.* 60-61 (2004), pp. 17–139 (cit. on p. 170).

[Pow99] John Power. “Enriched Lawvere Theories”. In: *Theory and Applications of Categories* 6.7 (1999), pp. 83–93 (cit. on p. 250).

[PP02] Gordon D. Plotkin and John Power. “Notions of Computation Determine Monads”. In: *FoSSaCS*. Vol. 2303. Lecture Notes in Computer Science. Springer, 2002, pp. 342–356 (cit. on pp. 25, 169, 237, 238, 241).

[PR95] D. Pumplün and Helmut Röhrl. “Convexity theories IV. Klein-Hilbert parts in convex modules”. In: *Appl. Categorical Struct.* 3.2 (1995), pp. 173–200 (cit. on p. 169).

[PRW21] Damien Pous, Jurriaan Rot, and Jana Wagemaker. “On Tools for Completeness of Kleene Algebra with Hypotheses”. In: *RAMICS*. 2021, pp. 378–395 (cit. on pp. 167, 245).

[PW22] Damien Pous and Jana Wagemaker. “Completeness Theorems for Kleene Algebra with Top”. In: *CONCUR*. 2022, 26:1–26:18 (cit. on p. 167).

[PZ23] Robin Piedeleu and Fabio Zanasi. “A Finite Axiomatisation of Finite-State Automata Using String Diagrams”. In: *Log. Methods Comput. Sci.* 19.1 (2023) (cit. on p. 250).

[Red64] V. N. Redko. “On defining relations for the algebra of regular events”. In: *Ukrainskii Matematicheskii Zhurnal* 16 (1 1964), pp. 120–126 (cit. on pp. 20, 250).

[Ree02] J. Rees. *Fizz Buzz: 101 Spoken Numeracy Games - Ideal for Mental Maths*. 101 Spoken Numeracy Games - Ideal for Mental Maths. LDA, 2002 (cit. on p. 79).

[RHE22] Aloïs Rosset, Helle Hvid Hansen, and Jörg Endrullis. “Algebraic Presentation of Semifree Monads”. In: *CMCS*. Vol. 13225. Lecture Notes in Computer Science. Springer, 2022, pp. 110–132 (cit. on pp. 175, 176).

[Ros00] Brian J. Ross. “Probabilistic Pattern Matching and the Evolution of Stochastic Regular Expressions”. In: *Appl. Intell.* 13.3 (2000), pp. 285–300 (cit. on p. 20).

[Róż+23] Wojciech Różowski, Tobias Kappé, Dexter Kozen, Todd Schmid, and Alexandra Silva. *Probabilistic Guarded KAT Modulo Bisimilarity: Completeness and Complexity*. 2023. arXiv: 2305.01755 [cs.LO] (cit. on pp. 8, 226, 227).

[Rut00] Jan J. M. M. Rutten. "Universal coalgebra: a theory of systems". In: *Theoretical Computer Science* 249.1 (2000), pp. 3–80 (cit. on pp. 23, 27, 33, 45, 47, 49, 60, 68, 69, 173, 188, 213, 216, 236, 242).

[Rut03] Jan J. M. M. Rutten. "Behavioural differential equations: a coinductive calculus of streams, automata, and power series". In: *Theoretical Computer Science* 308 (2003), pp. 1–53 (cit. on p. 53).

[Rut98] Jan J. M. M. Rutten. "Automata and Coinduction (An Exercise in Coalgebra)". In: *CONCUR*. Vol. 1466. Lecture Notes in Computer Science. Springer, 1998, pp. 194–218 (cit. on pp. 23, 101, 190).

[Sal66] Arto Salomaa. "Two Complete Axiom Systems for the Algebra of Regular Events". In: *J. ACM* 13.1 (1966), pp. 158–169 (cit. on pp. 20, 31, 32, 34, 35, 39, 55, 76, 98, 165, 171, 172, 218, 238, 241).

[SBR10] Alexandra Silva, Marcello M. Bonsangue, and Jan J. M. M. Rutten. "Non-Deterministic Kleene Coalgebras". In: *Log. Methods Comput. Sci.* 6.3 (2010) (cit. on pp. 66, 67).

[Sch+21] Todd Schmid, Tobias Kappé, Dexter Kozen, and Alexandra Silva. "Guarded Kleene Algebra with Tests: Coequations, Coinduction, and Completeness". In: *ICALP*. Vol. 198. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2021, 142:1–142:14 (cit. on pp. 8, 27, 28, 76, 78, 87, 93, 96, 112, 172, 225).

[Sch+22] Todd Schmid, Wojciech Rozowski, Jurriaan Rot, and Alexandra Silva. "Processes Parametrised by an Algebraic Theory". In: *ICALP*. Vol. 229. LIPIcs. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2022, 132:1–132:20 (cit. on pp. 27, 29, 76, 165).

[Sch22a] Todd Schmid. "A (Co)Algebraic Framework for Ordered Processes". In: *CoRR* abs/2209.00634 (2022). arXiv: 2209.00634 (cit. on pp. 27, 29, 166, 250).

[Sch22b] Todd Schmid. “Presenting with Quantitative Inequational Theories”. In: *CoRR* abs/2207.11629 (2022). arXiv: 2207.11629 (cit. on p. 181).

[Sil10] Alexandra Silva. “Kleene coalgebra”. PhD thesis. Radboud University, 2010 (cit. on pp. 26, 28, 32–34, 52, 66, 67, 72, 239).

[Smo+17] Steffen Smolka, Praveen Kumar, Nate Foster, Dexter Kozen, and Alexandra Silva. “Cantor meets Scott: semantic foundations for probabilistic networks”. In: *POPL*. 2017, pp. 557–571 (cit. on pp. 75, 168).

[Smo+19] Steffen Smolka, Praveen Kumar, David M. Kahn, Nate Foster, Justin Hsu, Dexter Kozen, and Alexandra Silva. “Scalable verification of probabilistic networks”. In: *PLDI*. 2019, pp. 190–203 (cit. on p. 75).

[Smo+20] Steffen Smolka, Nate Foster, Justin Hsu, Tobias Kappé, Dexter Kozen, and Alexandra Silva. “Guarded Kleene algebra with tests: verification of uninterpreted programs in nearly linear time”. In: *Proc. ACM Program. Lang.* 4.POPL (2020), 61:1–61:28 (cit. on pp. 8, 19, 21–23, 75–77, 80–84, 87, 90, 92, 96, 98–100, 104, 112, 155, 165, 167, 172, 224, 225, 242).

[SP00] Alex K. Simpson and Gordon D. Plotkin. “Complete Axioms for Categorical Fixed-Point Operators”. In: *15th Annual IEEE Symposium on Logic in Computer Science, Santa Barbara, California, USA, June 26-29, 2000*. IEEE Computer Society, 2000, pp. 30–41 (cit. on p. 251).

[SRS21] Todd Schmid, Jurriaan Rot, and Alexandra Silva. “On Star Expressions and Coalgebraic Completeness Theorems”. In: *MFPS*. Vol. 351. EPTCS. 2021, pp. 242–259 (cit. on pp. 27, 28, 34).

[SS00] Eugene W. Stark and Scott A. Smolka. “A complete axiom system for finite-state probabilistic processes”. In: *Proof, Language, and Interaction*. The MIT Press, 2000, pp. 571–596 (cit. on pp. 8, 237, 246).

[Sto49] Marshall Harvey Stone. “Postulates for the barycentric calculus”. In: *Annali di Matematica Pura ed Applicata* 29.1 (1949), pp. 25–30 (cit. on p. 169).

[Świ74] Tadeusz Świrszcz. “Monadic Functors and Convexity”. In: *Bulletin de L’Académie Polonaise des Sciences*. Serie de math., astr., and phys. XXII.1 (1974), pp. 39–42 (cit. on p. 169).

[Tar75] Robert Endre Tarjan. “Efficiency of a Good But Not Linear Set Union Algorithm”. In: *J. ACM* 22.2 (1975), pp. 215–225 (cit. on p. 76).

[TF06] Toshinori Takai and Hitoshi Furusawa. “Monodic Tree Kleene Algebra”. In: *RelMICS/AKA*. 2006, pp. 402–416 (cit. on p. 166).

[Tho68] Ken Thompson. “Programming Techniques: Regular Expression Search Algorithm”. In: *Commun. ACM* 11.6 (June 1968), pp. 419–422 (cit. on p. 36).

[TP97] Daniele Turi and Gordon D. Plotkin. “Towards a Mathematical Operational Semantics”. In: *LICS*. IEEE Computer Society, 1997, pp. 280–291 (cit. on p. 246).

[TR98] Daniele Turi and Jan J. M. M. Rutten. “On the Foundations of Final Coalgebra Semantics: Non-Well-Founded Sets, Partial Orders, Metric Spaces”. In: vol. 8. 5. 1998, pp. 447–480 (cit. on p. 188).

[VW06] Daniele Varacca and Glynn Winskel. “Distributing probability over non-determinism”. In: *Mathematical Structures in Computer Science* 16.1 (2006), pp. 87–113 (cit. on p. 181).

[Wag+19] Jana Wagemaker, Marcello M. Bonsangue, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. “Completeness and Incompleteness of Synchronous Kleene Algebra”. In: *MPC*. 2019, pp. 385–413 (cit. on pp. 20, 165).

[Wag+20] Jana Wagemaker, Paul Brunet, Simon Docherty, Tobias Kappé, Jurriaan Rot, and Alexandra Silva. “Partially Observable Concurrent Kleene Algebra”. In: *CONCUR*. 2020, 20:1–20:22 (cit. on pp. 165, 167).

[ZSS22] Stefan Zetzsche, Alexandra Silva, and Matteo Sammartino. “Guarded Kleene Algebra with Tests: Automata Learning”. In: *MFPS*. 2022 (cit. on pp. 76, 225).

## Appendix A

# ACF Extends GKAT

In this technical appendix, I show that the general framework that I have laid out in the thesis instantiates to the example of GKAT from the literature. More specifically, I am going to show that the star fragment of the algebra of control flows ACF, the effectful process calculus obtained from the theory of guarded algebra (Example 4.2.7), is equivalent to bisimulation GKAT.

## A.1 GKAT is an Effectful Process Algebra

Write *StExp* for the star fragment syntax in Example 4.5.1, and *GExp* for the expression language in Equation (3.1) from Chapter 3. We define two translations,

$$
\begin{aligned}
&stg\colon StExp \to GExp && gts\colon GExp \to StExp\\
&\quad stg(0) = 0 && \quad gts(p) = p\\
&\quad stg(1) = 1 && \quad gts(b) = 1 +_{\{\alpha \le_{\mathsf{BA}} b\}} 0\\
&\quad stg(p) = p && gts(e +_b f) = gts(e) +_{\{\alpha \le_{\mathsf{BA}} b\}} gts(f)\\
&stg(e +_A f) = stg(e) +_{\bigvee A} stg(f) && \quad gts(ef) = gts(e) \cdot gts(f)\\
&\quad stg(ef) = stg(e) \cdot stg(f) && gts(e^{(b)}) = gts(e)^{(\{\alpha \le_{\mathsf{BA}} b\})}\\
&stg(e^{(A)}) = stg(e)^{(\bigvee A)}
\end{aligned}
$$

Notice that $gts \circ stg = \mathrm{id}_{StExp}$. Furthermore, since

$$\mathsf{GKAT}_0 \vdash 1 +_b 0 \overset{\text{(G9,DM)}}{=} b +_b 0 = b +_b \bar{b}b = b +_b b \overset{\text{(G1)}}{=} b$$

an easy inductive argument shows that for any $g \in GExp$, $\mathsf{GKAT}_0 \vdash g = stg(gts(g))$.

We are going to show that both translations preserve bisimilarity and provable

$$(\mathsf{EQ}^*1)\quad 1e = e1 = e \qquad (\mathsf{EQ}^*3)\quad e_1(e_2e_3) = (e_1e_2)e_3$$
$$(\mathsf{EQ}^*2)\quad ce = c \qquad (\mathsf{EQ}^*4)\quad e_1^{(\sigma)}e_2 = e_1e_1^{(\sigma)}e_2 +_\sigma e_2$$
$$(\mathsf{EQ}^*5)\quad \frac{(\forall i \le n)\ g_i \text{ is guarded}}{p(1,\vec{g})^{(\sigma)} = \mathsf{fp}\,\underline{u}\ (p(\underline{u},\vec{g}p(1,\vec{g})^{(\sigma)}) +_\sigma 1)}$$
$$(\mathsf{EQ}^*6)\quad \frac{g = eg +_\sigma f \quad e \text{ is guarded}}{g = e^{(\sigma)}f}$$

**Figure A.1:** The axioms of $\mathsf{EQ}^*$. Here $c \in S_0$, $\sigma \in S_2$, and $p(x,\vec{y}) \in S^*X$.

equivalence. Here on out, we are going to identify a subset $A \subseteq At$ with its Boolean test $\bigvee A$, and a Boolean test $b \in BExp$ with its set of atoms $\{\alpha \le_{\mathsf{BA}} b\}$ whenever convenient. The argument for bisimilarity is straightforward: It suffices to show the following.

**Lemma A.1.1.** *The translation maps stg and gts are $L_M$-coalgebra homomorphisms.*

The proof of the above lemma is an easy induction on expressions in both cases.

**Translation preserves provable equivalence**

In order to see that the star fragment $\mathsf{GA}^*$ of ACF (Example 4.2.7) is equivalent to bisimulation GKAT, we show that proofs in $\mathsf{GA}^*$ can be replicated in bisimulation GKAT, and vice versa.

We begin with the following lemma.

**Lemma A.1.2.** *For any $e \in StExp$ and $g \in GExp$, $e$ is guarded if and only if $E(stg(e)) =_{\mathsf{BA}} 0$ and $E(g) =_{\mathsf{BA}} 0$ iff $gts(g)$ is guarded.*

The proof of the above lemma is straightforward, so we move on to the soundness of the forward translation.

The next lemma tells us that every equivalence provable in the star fragment $\mathsf{GA}^*$ of ACF can be proven in bisimulation GKAT.

**Lemma A.1.3.** *Let $e, f \in StExp$. If $\mathsf{GA}^* \vdash e = f$, then $\mathsf{GKAT}_0 \vdash stg(e) = stg(f)$.*

*Proof.* By induction on the proof of $\mathsf{GA}^* \vdash e = f$. Since $stg$ is an algebra homomorphism, if $\mathsf{GA}^*1 + \cdots + \mathsf{GA}^*4 + \mathsf{GA} \vdash e = f$, then $\mathsf{GKAT}_0 \vdash stg(e) = stg(f)$. It therefore suffices to check the axioms $\mathsf{GA}^*5$ and $\mathsf{GA}^*6$.

For $\mathsf{GA}^*5$, let $g_1,\dots,g_n$ be guarded, and let $p(u,\vec{x}) \in S^*_{gsl}X$. Then $stg(g_1),\dots,stg(g_n)$ are guarded in $GExp$, and $\mathsf{fp}\,\underline{u}\ (p(\underline{u},stg(\vec{g}))+_b 1) = p(0,stg(\vec{g}))+_b 1$. Now, there is a $c \subseteq At$ such that for some $q(\vec{x})$, $\mathsf{GA} \vdash p(1,stg(\vec{g})) = 1 +_c q(stg(\vec{g}))$. Hence,

$$\begin{aligned}
\mathsf{GKAT}_0 \vdash stg(p(1,\vec{g})^{(b)}) &= p(1,stg(\vec{g}))^{(b)} \\
&= (1+_c q(stg(\vec{g})))^{(b)} \\
&= (0+_c q(stg(\vec{g})))^{(b)} \\
&= (p(0,stg(\vec{g})))^{(b)} \\
&= p(0,stg(\vec{g}))\cdot(p(0,stg(\vec{g})))^{(b)} +_b 1 \\
&= p(0,stg(\vec{g})\cdot(p(0,stg(\vec{g})))^{(b)}) +_b 1 \\
&= \mathsf{fp}\,\underline{u}\ (p(\underline{u},stg(\vec{g})\cdot(p(0,stg(\vec{g})))^{(b)}) +_b 1) \\
&= \mathsf{fp}\,\underline{u}\ (p(\underline{u},stg(\vec{g})\cdot(p(1,stg(\vec{g})))^{(b)}) +_b 1) \\
&= stg(\mathsf{fp}\,\underline{u}\ (p(\underline{u},\vec{g}p(1,\vec{g}))^{(b)}) +_b 1)
\end{aligned}$$

Since $E(stg(e)) =_{\mathsf{BA}} 0$ when $e$ is guarded, if $\mathsf{GA}^* \vdash g = eg +_b f$ and

$$\mathsf{GKAT}_0 \vdash stg(g) = stg(eg+_b f) = stg(e)\cdot stg(g) +_b stg(f)$$

then by $(\mathsf{GA}^*6)$ and the induction hypothesis,

$$\mathsf{GA}^* \vdash stg(g) = stg(e)^{(b)}\cdot stg(f) = stg(e^{(b)}f) \qquad \square$$

The next lemma tells us that every equivalence provable in bisimulation GKAT can be proven in the star fragment $\mathsf{GA}^*$ of ACF.

**Lemma A.1.4.** *Let $e,f \in GExp$. If $\mathsf{GKAT}_0 \vdash e = f$, then $\mathsf{GA}^* \vdash gts(e) = gts(f)$.*

*Proof.* Again, by induction on the proof of $\mathsf{GKAT}_0 \vdash e = f$. If $e = b \in BExp$ and $f = c \in BExp$ and $\mathsf{BA} \vdash b = c$, then $\mathsf{GA}^* \vdash 1+_b 0 = 1+_c 0$ because $\mathcal{P}At$ is a Boolean algebra (technically, $gts(b) = 1+_{\{\alpha\in At \mid \alpha \le_{\mathsf{BA}} b\}} 0$). If $\vdash e = f$ follows from one application of any rules in the Guarded Union column of Figure 3.5, then $\mathsf{GA} \vdash e = f$. Also, $(\mathsf{GA}^*1)$-$(\mathsf{GA}^*4)$ coincide precisely with sequencing and loop axioms of GKAT.

Now let's check the equation $(e+_c 1)^{(b)} = (ce)^{(b)}$ is preserved by $gts$. This amounts

to checking that

$$\mathsf{GA}^* \vdash (gts(e) +_c 1)^{(b)} = (gts(e) +_c 0)^{(b)}$$

Unsurprisingly, $\mathsf{GA}^*$ satisfies an analogue of the fundamental theorem of GKAT, so we know that there is a $d \in BExp$ such that for some guarded $g \in StExp$, $\mathsf{GA}^* \vdash gts(e) = g +_d 1$. Hence, on the one hand,

$$\begin{aligned}
\mathsf{GA}^* \vdash (gts(e) +_c 1)^{(b)} &= ((g +_d 1) +_c 1)^{(b)} \\
&= ((g((g +_d 1) +_c 1)^{(b)} +_d 0) +_c 0) +_b 1 && (\mathsf{GA}^*5) \\
&= ((g +_d 0) +_c 0)((g +_d 1) +_c 1)^{(b)} +_b 1 \\
&= ((g +_d 0) +_c 0)^{(b)} && (\mathsf{GA}^*6)
\end{aligned}$$

And on the other hand,

$$\begin{aligned}
\mathsf{GA}^* \vdash ((g +_d 1) +_c 0)^{(b)} &= ((g((g +_d 1) +_c 0)^{(b)} +_d 0) +_c 0) +_b 1 && (\mathsf{GA}^*5) \\
&= ((g +_d 0) +_c 0)((g +_d 1) +_c 0)^{(b)} +_b 1 \\
&= ((g +_d 0) +_c 0)^{(b)} && (\mathsf{GA}^*6)
\end{aligned}$$

It follows that

$$\begin{aligned}
\mathsf{GA}^* \vdash (gts(e) +_c 1)^{(b)} &= ((g +_d 1) +_c 1)^{(b)} \\
&= ((g +_d 0) +_c 0)^{(b)} \\
&= ((g +_d 1) +_c 0)^{(b)} \\
&= (gts(e) +_c 0)^{(b)}
\end{aligned}$$

as desired. Finally, if $\mathsf{GKAT}_0 \vdash g = e \cdot g +_b f$ and

$$\mathsf{GA}^* \vdash gts(g) = gts(e \cdot g +_b f) = gts(e) \cdot gts(g) +_c gts(f)$$

and $e$ is guarded, then since $E(gts(e)) =_{\mathsf{BA}} 0$, by $(\mathsf{GA}^*5)$,

$$\mathsf{GA}^* \vdash gts(g) = gts(e)^{(c)} gts(f) = gts(e^{(b)} \cdot f) \qquad \square$$

This concludes the proof of equivalence between $\mathsf{GA}^*$ and bisimulation GKAT,

because

$$\begin{aligned} \mathsf{GA}^* \vdash e = f &\implies \mathsf{GKAT}_0 \vdash \mathrm{stg}(e) = \mathrm{stg}(f) \\ &\implies \mathsf{GA}^* \vdash \mathit{gts} \circ \mathrm{stg}(e) = \mathit{gts} \circ \mathrm{stg}(f) \\ &\iff \mathsf{GA}^* \vdash e = f \end{aligned}$$

It follows that $\mathit{stg}\colon \mathit{StExp}/\mathsf{GA}^* \to \mathit{GExp}/\mathsf{GKAT}_0$, the corresponding map between congruence classes, is a bijection. Furthermore, $\mathit{stg}$ preserves the algebraic operations of $\mathsf{GA}^*$. This implies that $\mathit{StExp}/\mathsf{GA}^*$ and $\mathit{GExp}/\mathsf{GKAT}_0$ are isomorphic as $\Omega$-algebras, where $\Omega$ is algebraic signature of $\mathsf{GA}^*$. It is in this sense that bisimulation GKAT is equivalent to $\mathsf{GA}^*$.