Session-Typed Ordered Logical Specifications

Henry DeYoung

CMU-CS-??-???

September 29, 2020

School of Computer Science
Carnegie Mellon University
Pittsburgh, PA 15213

Thesis Committee:
Frank Pfenning, Chair
Iliano Cervesato
Robert Harper
André Platzer
Simon Gay, University of Glasgow
Carsten Schürmann, IT University of Copenhagen

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy.

This research was sponsored by National Science Foundation award number: 1718276; by a National Science Foundation Graduate Research Fellowship award; and by a Google Lime Scholarship award.

The views and conclusions contained in this document are those of the author and should not be interpreted as representing the official policies, either expressed or implied, of any sponsoring institution, the U.S. government or any other entity.

Keywords: concurrency, bisimilarity, session types, proof construction, proof reduction, ordered logic, singleton logic

Abstract

Concurrent systems are ubiquitous, but notoriously difficult to get right: subtle races and deadlocks can lurk even in the most extensively tested of systems. In a quest to tame concurrency, researchers have successfully applied the principle of *computation as deduction* to concurrency in two distinct ways: *concurrency as proof reduction* and *concurrency as proof construction*. These two approaches to concurrency have complementary advantages, with the proof-construction approach excelling at global specification of a system's dynamics, while the proof-reduction approach is best suited to implementation of the processes that comprise the system.

This document explores the relationship between these two different proof-theoretic characterizations of concurrency. To focus on the essential aspects of their relationship, the exploration is carried out in the context of concurrent systems that have chain topologies. From a proof-construction perspective, chain topologies arise from ordered logic; from a proof-reduction perspective, they arise from *singleton logic*, a variant of ordered logic that restricts sequents to have exactly one antecedent.

In this context, a rewriting framework is systematically derived from the ordered sequent calculus, and a *message-passing* fragment of that rewriting framework is identified. String rewriting specifications of concurrent systems can be *choreographed* into this fragment, and the fragment supports a notion of bisimilarity. Along the way, we also uncover a *semi-axiomatic sequent calculus* for singleton logic, which blends a standard sequent calculus with axiomatic aspects of Hilbert systems, and we then establish a correspondence between semi-axiomatic proof normalization and asynchronous message-passing communication. Ultimately, the message-passing processes can be faithfully embedded within the message-passing ordered rewriting framework in a bisimilar way. Perhaps surprising is that, because the embedding is left-invertible, we can also identify fairly broad conditions under which local, process implementations can be *extracted* from global, message-passing ordered rewriting specifications.

Contents

1
Introduction

With the increasingly complex, distributed nature of today's software systems, concurrency is ubiquitous. Concurrency facilitates distributed computation by structuring systems as nondeterministic compositions of simpler subsystems. But, being nondeterministic, concurrent systems are notoriously difficult to get right: subtle races and deadlocks can be lurking in even the most extensively tested of systems.

At the same time, decades of research into connections between proof theory and programming languages – beginning with Curry's observation that simplification of axiomatic proofs corresponds to combinatory reduction,[1] and notably continuing with Howard's discovery of an isomorphism between intuitionistic natural deduction and the simply-typed λ-calculus[2] – have firmly established the principle of *computation as deduction* as the gold standard for clear, expressive, and provably correct programs. Computation-as-deduction interpretations of intuitionistic logic, for example, are the foundations for both the typed functional[3] and logic[4] programming paradigms.

[1] Curry 1934.

[2] Howard 1969.

[3] Martin-Löf 1982.

[4] Miller et al. 1991; Andreoli 1992.

The computation-as-deduction idea has also been successfully applied to concurrent programming, originating from Girard's suggestion of possible connections between linear logic and concurrency.[5] These research efforts have been directed along two different proof-theoretic paths: *concurrency as proof reduction* and *concurrency as proof construction*.

[5] Girard 1987.

Under a concurrency-as-proof-reduction view, processes are mapped to linear sequent proofs or proof nets. Proof reduction, in the form of cut elimination, thus corresponds to concurrent, message-passing communication. This view was pioneered by Abramsky,[6] further pursued by Bellin and Scott,[7] and later extended to a true Curry–Howard isomorphism with the intuitionistic linear sequent calculus by Caires and Pfenning with Toninho.[8] Under this isomorphism, propositions are types – specifically, binary *session types*[9] that describe the interaction protocol to which a process adheres. And concurrency arises when the various interleavings of independent proof reductions are treated indistinguishably.

[6] Abramsky 1993.

[7] Bellin and Scott 1994.

[8] Caires and Pfenning 2010; Caires, Pfenning, and Toninho 2012, 2016.

[9] Honda 1993.

Under the other, concurrency-as-proof-construction view, computation is the act of building a partial, cut-free proof by nondeterministically applying

the inference rules one by one. In our setting, this proof construction tack is perhaps best encapsulated by the *process-as-formula* encoding, in which process constructors are mapped to linear logical connectives.[10] Concurrency is then manifested by the permutability of inference rules within a partial proof.

[10] Miller 1992.

Despite the various research efforts into proof-reduction and proof-construction approaches to concurrency as deduction, it appears that relatively little research on the relationship between the two approaches exists in the literature, with an article by Cervesato and Scedrov being the notable instance.[11] In this document, we undertake a further study of the relationship between proof reduction and proof construction.

[11] Cervesato and Scedrov 2009.

Specifically, we begin by noticing that each of these two approaches to concurrency as deduction has its own strengths and weaknesses. The proof-reduction approach excels at implementation. Under Caires, Pfenning, and Toninho's isomorphism, proofs are immediately and directly well-typed process implementations.[12] Properties of cut elimination also ensure that the proof-reduction approach enjoys session-type preservation and progress theorems – well-typed processes will never deadlock.[13]

[12] Toninho et al. 2013; Griffith 2016.

[13] But they can livelock or diverge in the presence of recursive types, unless those recursive types are strictly coinductive (Derakhshan and Pfenning 2020).

However, the proof-reduction approach does have its weaknesses. Owing to the binary structure of the cut elimination procedure, the proof-reduction approach lends itself more naturally to synchronous communication, whereas asynchronous communication is often more consistent with programming practice and more directly realizable. Also, because processes are generally long-running or non-terminating, recursion is even more important than in functional programming, and it is not obvious how to incorporate recursion in a logically justified way within the session type isomorphism.[14]

[14] Recursion can be incorporated in functional languages in a logically justified way, using inductive and coinductive types (Mendler 1987). Derakhshan and Pfenning (2020) and Somayyajula and Pfenning (2020) are currently investigating logically justified inductive and coinductive session types.

In contrast to proof reduction, the proof-construction approach excels at the task of specifying the behavior of concurrent systems. It gives a more global description of the system by focusing on the interactions between processes, without prescribing how those interactions occur. Because computation is captured by the construction of a partial cut-free proof and each inference rule has a single principal formula, this approach reflects *a*synchronous communication. Moreover, recursion is relatively easy to incorporate because the logical aspects of proof construction are confined to hypothetical derivability and partial proofs, rather than provability and complete proofs.

On the flip side, however, it is not obvious how to extract well-behaved, local process implementations from these global specifications. And, being untyped, the proof-construction approach does not enjoy type preservation and progress theorems like the concurrency-as-proof-reduction view does – computation can easily deadlock or livelock when proof construction fails or diverges, respectively.

In short, the strengths and weaknesses of the proof-reduction and proof-construction approaches are almost exactly opposite each other. To gain the

strengths of both approaches without the weaknesses of either, we would like to identify a kind of intersection between concurrency as proof reduction and concurrency as proof construction. That is, we want to identify fragments of proof reduction and proof construction that can be put in bijective correspondence. On these fragments, we will be able to give global specifications of concurrent systems, while still being able to extract, or project, local implementations from them.

To focus our attention on the essential aspects of and relationship between the proof-reduction and proof-construction approaches to concurrency as deduction, we will limit our investigation in this document to processes arranged in chain topologies, as opposed to the more general tree topologies that Caires, Pfenning, and Toninho's isomorphism supports. Chain topologies allow us to treat channels namelessly – each process has exactly two channels, one with its unique left-hand neighbor and one with its right-hand neighbor – which lifts a large but inessential notational burden.

In both the proof-reduction and proof-construction characterizations of concurrency, chain topologies exist as logically motivated fragments of the general case. From the proof-construction perspective, chain topologies arise from (modality-free) ordered logic,[15] an extension of the Lambek calculus;[16] from the proof-reduction perspective, chain topologies will arise from *singleton logic*,[17] an *a*structural fragment of ordered and linear logics in which sequents have exactly one antecedent and one consequent.

[15] Lambek 1961; Abrusci 1990; Kanazawa 1992; Polakow and Pfenning 1999b.

[16] Lambek 1958.

[17] Santocanale 2002; Fortier and Santocanale 2013.

Thus, the remainder of this document serves to establish the following thesis statement.

> *Session types form a bridge between distinct notions of concurrency in computational interpretations of singleton and ordered logics based on proof reduction, on one hand, and proof construction, on the other hand.*

We also conjecture that the results contained in this document can be generalized in a relatively straightforward way to tree topologies. This will introduce a moderately large notational overhead, but should present no conceptual difficulties. The singleton logic used for proof-reduction-as-concurrency would be replaced with propositional linear logic, and the modality-free ordered logic used for proof-construction-as-concurrency would be replaced with first-order linear logic.

1.1 *Overview*

In this section, we provide a high-level overview of this document.

Part I reviews some necessary background information, namely definitions of finite automata (chapter 2) and a sequent calculus presentation of ordered logic[18] (chapter 3). The reader who has some familiarity with these topics should feel free to skim or skip these chapters, returning to them as needed.

[18] Polakow and Pfenning 1999b.

1.1.1 *Concurrency as proof construction*

Part II then delves into a proof-construction approach to concurrency, beginning in chapter 4 with a review of a string rewriting framework for specifying the dynamics of concurrent systems. Specifically, string rewriting can be used for systems whose components are arranged into a chain topology and have a monoidal structure. Because disjoint segments of a string may be rewritten independently, concurrency arises when the various interleavings of these independent rewritings are treated as equivalent. String rewriting is an instance of multiset rewriting, so these ideas are not new, but are applied in a new setting.[19] Chapter 4 closes by introducing specifications for two systems that will be used as running examples throughout this document: nondeterministic finite automata (section 4.2) and binary counters (section 4.3).

[19] Meseguer 1992.

Part II purports to give a proof-construction approach to concurrency, but string rewriting, while indeed a framework for concurrency, is not obviously connected to proof construction. For that, chapter 5 turns our attention toward the Lambek calculus and ordered logic.

Implicit in Lambek's calculus for categorial grammars is a notion of rewriting for free residuated monoids.[20] Chapter 5 presents *ordered rewriting*, a related rewriting framework for free residuated lattices, which we will derive from the ordered sequent calculus. As we will see, the sequent calculus's left rules share a large amount of boilerplate – only very little of each left rule is devoted to the primary task of decomposing the principal proposition. In response, we argue for a refactoring of the ordered sequent calculus, introducing a new judgment to decouple decomposition from the surrounding boilerplate (section 5.1.1). Ordered rewriting is then exactly the decomposition-centered fragment of the refactored sequent calculus (section 5.1.2). To the best of our knowledge, our refactoring of the sequent calculus left rules appears to be a new way of deriving rewriting from existing proof theory.

[20] Lambek's calculus was later extended to free residuated lattices (Lambek 1961; Abrusci 1990; Kanazawa 1992).

As in string rewriting, ordered rewriting permits disjoint segments of the ordered context to be rewritten independently, and concurrency arises when the different interleavings of these independent rewritings are treated indistinguishably (section 5.1.3). And so ordered rewriting is the proof-construction characterization of concurrency that we were looking for.[21]

Chapter 5 closes by extending ordered rewriting with ideas from focusing[22] to better control the atomicity of individual rewriting steps (sections 5.2 and 5.3). The particular formulation we choose is Zeilberger's higher-order focusing.[23] In its focused form, ordered rewriting is closely related to the exponential-free fragment of Simmons's SLS framework.[24]

[21] Incidentally, this focused ordered rewriting framework is roughly what would be needed to combine the Ordered Logical Framework (Polakow 2001) with the Concurrent Logical Framework (Watkins et al. 2002).

[22] Andreoli 1992.

[23] Zeilberger 2008.

[24] Simmons 2012.

Despite being models of concurrency, both string rewriting and (focused) ordered rewriting lack an immediate notion of local execution. Both frameworks are global, state-transformation models of concurrency that presume the existence of a central conductor that orchestrates the computation. This

kind of global rewriting, although reasonable for concurrent specifications, will not map well to the locally executing process implementations that a proof-reduction approach to concurrency will eventually suggest in part III – the gap is simply too large.

Strongly inspired by the process-as-formula view of linear logic,[25] the first part of chapter 6 responds to this dilemma by presenting a local, message-passing interpretation of focused ordered rewriting (section 6.1): non-atomic propositions are viewed as static process expressions; ordered contexts, as runtime process configurations; and atomic propositions, as messages. Surprisingly, only two simple modifications of chapter 5's focused ordered rewriting framework are required to enable this message-passing interpretation.

[25] Miller 1992; Cervesato and Scedrov 2009.

With these modifications in place, a local interaction semantics for this message-passing interpretation of (focused) ordered rewriting can be given (section 6.2).

At this point, we also introduce coinductively defined negative propositions (section 6.1.3), described with definitions of the form $\hat{p}^- \triangleq A^-$. Traditionally, substructural frameworks introduce unbounded behavior by way of replication and the ! exponential.[26] However, surprisingly subtle interactions between replication and order make recursive definitions a much more suitable choice for bringing unbounded behavior to ordered rewriting in our setting.

[26] Polakow 2001; Watkins et al. 2002.

To SUMMARIZE what we have so far, ordered rewriting has provided an explanation of concurrency in terms of proof construction and, looking ahead to our ultimate goal, the message-passing interpretation identifies a fragment of (focused) ordered rewriting that admits a local, process-like model of concurrency. But how do the string rewriting specifications of chapter 4 fit into this puzzle?

The second part of chapter 6 answers that question by providing a procedure for *choreographing* string rewriting specifications into the message-passing interpretation of ordered rewriting (section 6.3). The basic idea is that the programmer will assign a role to each of the string rewriting symbols – a symbol becomes either an atom or a coinductively defined proposition. Thus, under the message-passing interpretation, each symbol becomes a message or a coinductively defined process. A choreography then consists of a role assignment together with definitions for each of its coinductively defined propositions.

Not all role assignments will lead to sensible choreographies, however. A sensible choreography is one in which the coinductive definitions admit rewritings that, up to the role assignment, exactly match the string rewriting specification's axioms. That is, a choreography is sensible if the role assignment serves as a bisimulation between the string rewriting specification and the message-passing choreography.[27] Depending on the particular role assignment, it is possible that no such set of definitions exists.

[27] This is the first appearance of a notion of bisimilarity in this document. Bisimilarity in its various guises will be a recurring theme throughout this document.

Section 6.3.1 describes, informally, the conditions under which a given role assignment fails to yield a sensible choreography. Then, in section 6.3.2, we present a procedure for constructing a solution if one exists. The algorithm is formulated as a judgment on role assignments and string rewriting specifications, and we prove that when a solution exists, the role assignment is indeed a bisimulation between the string rewriting specification and its choreography.

Chapter 6 closes by examining choreographies for the binary counter (section 6.4) and NFAs (section 6.5). We show how to prove the end-to-end adequacy of these choreographies as a composition of the string rewriting specification's adequacy with the adequacy of the choreographing procedure.

In proving the adequacy of the NFA choreography, we find ourselves wishing for an equivalence on ordered contexts that is coarser than mere equality. So chapter 7 develops a notion of bisimilarity for the message-passing ordered rewriting framework. It is an observational equivalence in which atomic propositions are observable when they appear at the outside edges of an ordered context, but all other propositions are opaque. Our ordered rewriting bisimilarity is related to Deng et al.'s contextual preorder for linear logic,[28] although differing in its formulation (as well as the underlying logic and its structural rules).

[28] Deng et al. 2016.

The definition of ordered rewriting bisimilarity is suitable for directly proving that two contexts are *not* bisimilar, but it is difficult to directly prove that two contexts *are* bisimilar. Therefore, section 7.1.1 presents a sound, and surprisingly complete, proof technique for ordered rewriting bisimilarity that is reminiscent of labeled bisimilarity from the π-calculus[29] and Deng et al.'s simulation preorder.[30]

[29] Sangiorgi and Walker 2003.

[30] Deng et al. 2016.

Chapter 7 closes our discussion of bisimilarity and, more broadly, the proof-construction approach to concurrency with two examples of ordered rewriting bisimilarity in action: a proof that the NFA choreography preserves bisimilarity (section 7.2), and a proof that binary counters are bisimilar exactly when they have equal denotations (section 7.3).

1.1.2 *Concurrency as proof reduction*

Part III investigates a different proof-theoretic explanation of concurrency – concurrency as proof reduction.

Chapter 8 begins this investigation by reviewing *singleton logic*,[31] an *a*structural intuitionistic logic that exhibits many of the symmetries of classical logic by restricting sequents to have exactly one antecedent and one consequent.[32] Singleton sequents are thus $A \vdash C$, as opposed to the sequents $\Omega \vdash C$ found in ordered logic, for example. Section 8.2 verifies that singleton logic's sequent calculus satisfies cut and identity elimination, which ensure that singleton logic has a well-defined proof-theoretic semantics. It is quite surprising that such a severe restriction on the structure of sequents yields a well-defined

[31] Santocanale 2002; Fortier and Santocanale 2013.

[32] Fortier and Santocanale were originally motivated by categorical semantic concerns, more so than symmetries.

logic that will also prove in chapter 9 to be computationally useful.

Of course, sequent calculi are not the only way to present logics, with natural deduction and axiomatic systems being two notable alternatives. The chapter continues in section 8.3 by introducing a novel presentation of singleton logic – its *semi-axiomatic sequent calculus*. As suggested by its name, the semi-axiomatic sequent calculus blends the sequent calculus with axiomatic features. Its rules are the same as those of the sequent calculus, except that some rules[33] are replaced with axioms. At first glance, making such replacements might seem unmotivated – is it even possible to prove cut elimination for such a calculus?

[33] Specifically, all right rules for positive connectives and all left rules for negative connectives.

No, it is not possible to eliminate *all* cuts from semi-axiomatic proofs. But, interestingly, the cuts that remain are nevertheless well-behaved: they are analytic cuts that satisfy the subformula property. So, although cut elimination does not, strictly speaking, hold for the semi-axiomatic sequent calculus, a proof normalization result based on cut reduction does hold, as shown in section 8.3.2. Key to this normalization procedure are several novel *associative cut reductions* and a slightly unusual justification for their termination.

The principal cut reductions that appear in semi-axiomatic proof normalization are also notable. Because axioms hold such a prominent position in the calculus, none of these principal reductions carry over cuts – only one of the cut's two constituent proofs contributing to the reduced result. In this way, the principal cut reductions are reminiscent of asynchronous message-passing communication, an observation which will later be crucial.

The essential ideas behind the semi-axiomatic calculus appear to be widely applicable, going beyond singleton logic. Follow-up work with Pfenning and Pruiksma has extended the concept of semi-axiomatic sequent calculi to intuitionistic propositional logic, where the calculus yields an isomorphism with shared memory concurrency.[34] We further conjecture that semi-axiomatic sequent calculi exist for all intuitionistic logics with sequent calculi that admit cut elimination, including linear logic and ordered logic.

[34] DeYoung, Pfenning, and Pruiksma 2020.

Chapter 9, following up on the observation that the semi-axiomatic sequent calculus's principal cut reductions have the same structure as asynchronous message-passing communication, develops a session-typed process calculus from singleton logic's semi-axiomatic sequent calculus. Under this Curry–Howard interpretation, propositions correspond to session types that describe a process's behavior; proofs, to processes arranged in a chain topology; and proof reduction, to asynchronous message-passing communication between processes (section 9.1).

This is very closely related to Sill, the Curry–Howard interpretation of the intuitionistic linear sequent calculus as a session-typed π-calculus discovered by Caires, Pfenning, and Toninho,[35] but with two key differences. First, as previously alluded, singleton logic's single-antecedent restriction affects proof structure in such a way that the corresponding processes have a chain

[35] Caires and Pfenning 2010; Caires, Pfenning, and Toninho 2012, 2016.

topology, as opposed to the tree topology of Sill processes.[36] Second, and arguably more importantly, the proof reductions of the semi-axiomatic sequent calculus correspond to asynchronous message-passing communication, whereas Sill, being based on a standard sequent calculus, most naturally corresponds to synchronous communication.[37]

In section 9.2, coinductively defined types and processes are introduced to make unbounded computation possible. This takes the calculus outside of a true isomorphism, with the coinductive definitions being extralogical. But research by Derakhshan and Pfenning (2020) and Somayyajula and Pfenning (2020) on logical justifications for behaviorally inductive and coinductive session types could be adapted here to restore a true isomorphism.

Chapter 9 closes with some example programs. Process definitions are given for the binary counter (section 9.3.1); infinite-word sequential transducers (section 9.3.2), as a twist on the recurring NFA example; and Turing machines (section 9.3.3). In particular, the Turing machine example shows that, when combined with coinductive definitions, the computational interpretation of even a logic as slight and seemingly feeble as singleton logic can be Turing-complete.

[36] The idea of restricting processes to have a chain topology is also present in work by Dezani-Ciancaglini et al. (2014); see chapter 9.

[37] An earlier paper (DeYoung, Caires, et al. 2012) attempted to give an asynchronous interpretation of the intuitionistic linear sequent calculus, but, in hindsight, that work seems rather ad hoc and unsatisfactory when compared with the asynchronous interpretation of the semi-axiomatic sequent calculus.

1.1.3 *Relationship between proof construction and proof reduction*

Part IV studies the relationship between the two proof-theoretic characterizations of concurrency – concurrency as proof construction, on the one hand, as exemplified by the (focused) ordered rewriting and choreographies of part II; and concurrency as proof reduction, on the other hand, as exemplified by singleton logic's semi-axiomatic sequent calculus and the process chains of part III. Chapter 10 begins by formalizing the message-passing view of ordered rewriting by defining an embedding of session-typed process chains into ordered rewriting (section 10.1). This embedding serves as a bisimulation between process chains and ordered contexts – between concurrency as proof reduction and concurrency as proof construction.

The embedding is quite natural in two respects. First, it elegantly maps process constructors to ordered logical connectives, with process composition corresponding to ordered conjunction, for example. Second, as shown in section 10.1.4, when applied to the example processes from chapter 9, the embedding results in the same coinductively defined propositions as those used as choreographies in chapter 6.

Additionally, because the embedding is, syntactically speaking, an injective mapping, its left inverse provides a way to construct processes from a large subset of ordered propositions. Thus, in section 10.2, we use the embedding to reverse-engineer a session type system for ordered rewriting in which well-typed processes correspond to well-typed propositions, and vice versa.

The left-invertible embedding and session type system for ordered rewriting allows us to write global, ordered rewriting specifications of concurrent

systems and then extract local, process implementations from them, provided that the specifications are well-typed. We can have all of the advantages of global specifications, together with all of the advantages of local implementations.

Thus, our results can be seen as a proof-theoretic analogue of multiparty session types.[38] In multiparty session types, binary session types are generalized to conversations among several parties. Conversations in their entirety are specified using global session types, which can then be projected to binary session types for each pair of participants; these projections are close to implementations.

[38] Honda et al. 2008.

Intuitively, global types for multiparty sessions serve the same purpose as our choreographies: both describe the conversation as a whole. And, because both extract local information from a global description, the projection of local types from global types is related to our embedding of well-typed processes as choreographies. Moreover, our framework has the advantage of generating implementations directly from choreographies, whereas the multiparty session type discipline generates only local types that programmers must then implement.

1.1.4 *Concluding thoughts*

Chapter 10 provides final witness to the thesis stated earlier. At least for the well-typed fragment, proof construction and proof reduction are truly two sides of the same concurrent coin.

As described further in chapter 11, this document raises several avenues for future work. These include the obvious generalization of choreographies and the process embedding to proof-construction and proof-reduction notions of concurrency found in intuitionistic linear logic, an investigation of session-typed nondeterministic choice, a study of how the behaviorally inductive and coinductive types of Derakhshan and Pfenning[39] and sized types of Somayyajula and Pfenning[40] might apply to the proof-construction notion of concurrency, and an exploration of whether generative invariants[41] might relate to session types.

[39] Derakhshan and Pfenning 2020.

[40] Somayyajula and Pfenning 2020.

[41] Simmons 2012.

Part I

Preliminaries

2

Binary relations and automata

In this chapter, we review the definitions of alphabets, words, languages, and automata that we will use in the running examples throughout this document. Our definitions, though equivalent to the classical ones found in Hopcroft et al.'s text,[1] differ slightly, having been tuned for the particular applications in this document.

[1] Hopcroft et al. 2006.

First, however, we describe our notational conventions for binary relations.

2.1 *Binary relations*

In this and future chapters, we make significant use of binary relations of various kinds. These are often written in infix notation.

Given a binary relation $\mathcal{R}$, we write $\mathcal{R}^{-1}$ for its inverse. For relations written as arrows, such as $\longrightarrow$ and $\Longrightarrow$, we often instead express their inverses by writing the arrow in the other direction. For instance, $\Longleftarrow$ would be the inverse of $\Longrightarrow$, so that $y \Longleftarrow x$ exactly when $x \Longrightarrow y$.

We write the relational composition of $\mathcal{R}$ and $\mathcal{S}$ as juxtaposition, so that $x \mathcal{R}\mathcal{S} z$ holds exactly when there exists a y such that $x \mathcal{R} y$ and $y \mathcal{S} z$.

2.2 *Alphabets, words, and languages*

An alphabet Σ is simply a set of symbols, $a \in \Sigma$. A finite word w over the alphabet Σ is then a (possibly empty) finite sequence of symbols drawn from Σ; we denote the empty word by ϵ. The finite words form a free monoid under concatenation, with ϵ being the unit. We denote by Σ^* the set of all finite words over Σ.

It is also possible to construct infinite words. An infinite word over the alphabet Σ is a countably infinite sequence of letters drawn from Σ; we denote the set of all infinite words over Σ by Σ^ω. We also use Σ^∞ to denote the set of all words – finite or infinite – over the alphabet Σ; that is, $\Sigma^\infty = \Sigma^* \cup \Sigma^\omega$.

A language is a set of words. Depending on the context, it will be a subset of either Σ^∞, Σ^ω, or Σ^*.

2.3 *Nondeterministic and deterministic finite automata*

Definition 2.1. *A nondeterministic finite automaton (NFA)* over a finite input alphabet Σ is a triple $\mathcal{A} = (Q, \Delta, F)$ consisting of:

- a finite set of *states*, Q;
- a *transition function*, $\Delta\colon Q \times \Sigma \to \mathcal{P}(Q)$ such that $\Delta(q, a) \neq \emptyset$ for all states $q \in Q$ and input symbols $a \in \Sigma$; and
- a subset of *final states*, $F \subseteq Q$,

If $q' \in \Delta(q, a)$, then we say that q' is an *a-successor* of q and write $q \xrightarrow{a} q'$.[2]

The transition function Δ can be lifted to a relation involving finite input words: For each word $w = a_1 a_2 \cdots a_n \in \Sigma^*$, define a relation $\xrightarrow{w} \subseteq Q \times Q$ such that $q \xrightarrow{w} q'$ when $q = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q'$ for some sequence of states $q_0, q_1, \ldots, q_n \in Q$.

The NFA $\mathcal{A}$ accepts input word w from state q if there exists a state $q' \in Q$ such that $q \xrightarrow{w} q' \in F$; otherwise, the automaton rejects word w from state q. The language of all words accepted by automaton $\mathcal{A}$ from state q is denoted by $\mathcal{L}_{\mathcal{A}}(q)$.[3]

[2] The condition placed on Δ thus serves to ensure that, for all input symbols a, each state q has an a-successor – that is, that the NFA $\mathcal{A}$ cannot get stuck.

[3] We sometimes omit the subscript if the automaton is clear from the context.

Notice that, unlike the classical definition of NFAs, this definition does not fix an initial state for the automaton. This is because we will be primarily interested in the operational aspects of an NFA, rather than its linguistic aspects.

Example 2.1. As a concrete example, consider the NFA $\mathcal{A}_1$ over the input alphabet $\Sigma = \{a, b\}$ that is depicted in fig. 2.1. This NFA accepts, from state q_0, exactly those words that end with b. For comparison, the only word accepted from state q_1 is ϵ. This NFA is indeed nondeterministic, as both $q_0 \xrightarrow{b} q_0$ and $q_0 \xrightarrow{b} q_1$ hold. □

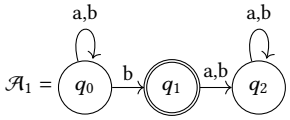


Figure 2.1: An NFA that accepts, from state q_0, exactly those words that end with b.

Definition 2.2. *A deterministic finite automaton (DFA)* over a finite input alphabet Σ is an NFA $\mathcal{A} = (Q, \Delta, F)$ over Σ in which $\Delta(q, a)$ is a singleton set for all states q and input symbols a. In this case, we write δ for the function from $Q \times \Sigma$ to Q that underlies Δ.

Example 2.2. Figure 2.2 depicts a DFA over the input alphabet $\Sigma = \{a, b\}$ that accepts, from state s_0, exactly those words that end with b. For comparison, the empty word ϵ, too, is accepted from the state s_1. □

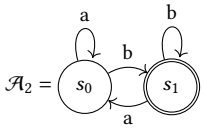


Figure 2.2: A DFA that accepts, from state s_0, exactly those words that end with b.

2.3.1 *NFA bisimilarity*

In later chapters, we will refer to a standard notion of bisimilarity for NFAs.

In general, two objects are bisimilar if they cannot be distinguished by an observer. Here, NFA bisimilarity is a relation on states, and the observer may provide an input word and observe whether the word is accepted or rejected by the given state.

Definition 2.3. Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over an input alphabet Σ. An *NFA bisimulation* on $\mathcal{A}$ is a *symmetric* binary relation on states, $\mathcal{R} \subseteq Q \times Q$, that satisfies the following conditions.

Input bisimulation If $q \mathrel{\mathcal{R}} \xrightarrow{a} s'$, then $q \xrightarrow{a} \mathrel{\mathcal{R}} s'$.
Finality bisimulation If $q \mathrel{\mathcal{R}} s \in F$, then $q \in F$.

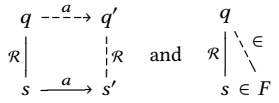


Figure 2.3: NFA bisimilarity, in diagrams

NFA bisimilarity on $\mathcal{A}$, written $\sim_{\mathcal{A}}$, is the largest bisimulation on $\mathcal{A}$. We will usually omit the subscript because the automaton is nearly always clear from context.

As a matter of convenience, NFA bisimilarity is defined on a single NFA. If we wish to discuss the bisimilarity of states from distinct NFAs, we can form the disjoint union of the two NFAs and work with its bisimilarity relation.

Bisimilarity is an equivalence relation.

Theorem 2.1. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over an input alphabet Σ. NFA bisimilarity on $\mathcal{A}$ is reflexive, symmetric, and transitive:*

Reflexivity *Equality on Q is a bisimulation on $\mathcal{A}$.*
Symmetry *If $\mathcal{R}$ is a bisimulation on $\mathcal{A}$, then so is $\mathcal{R}^{-1}$.*
Transitivity *If $\mathcal{R}$ and $\mathcal{S}$ are bisimulations on $\mathcal{A}$, then so is $\mathcal{RS}$.*

Proof. By the definition of bisimulation. □

The input bisimulation condition satisfied by an NFA bisimulation can be lifted to a condition on words, not just input symbols.

Theorem 2.2. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over an input alphabet Σ, and let $\mathcal{R}$ be an NFA bisimulation on $\mathcal{A}$. Then $q \mathrel{\mathcal{R}} \xrightarrow{w} s'$ implies $q \xrightarrow{w} \mathrel{\mathcal{R}} s'$.*

Proof. By induction over the structure of word w. □

NFA bisimilarity implies language equivalence.

Theorem 2.3. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over an input alphabet Σ. Then $q \sim s$ implies $\mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)$.*

Proof. Because bisimilarity is symmetric (theorem 2.1), it suffices to show that $q \sim s$ implies $\mathcal{L}_{\mathcal{A}}(q) \subseteq \mathcal{L}_{\mathcal{A}}(s)$. Let $q \in Q$ and $s \in Q$ be bisimilar states, and choose an arbitrary word w that is accepted from state q. By definition, $q \xrightarrow{w} q'_w \in F$ for some state q'_w. It follows from theorem 2.2 and the finality bisimulation condition that $s \xrightarrow{w} s'_w \in F$, for some state s'_w, and so w is also accepted from state s. □

But, because of nondeterminism, the converse does not hold.

False claim 2.4. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over an input alphabet Σ. Then $\mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)$ implies $q \sim s$.*

Counterexample. Choose the NFAs $\mathcal{A}_1$ and $\mathcal{A}_2$ given in figs. 2.1 and 2.2. Although the languages accepted by states q_0 and s_0 are the same, the two states are *not* bisimilar.

For the sake of deriving a contradiction, assume that $q_0 \sim s_0$. Because q_0 is one of the b-successors of q_0, it follows by the input bisimulation condition that $s_0 \xrightarrow{b} \sim q_0$. But s_1 is the unique b-successor of s_0, and so we may deduce that $s_1 \sim q_0$. Just as s_1 is a final state, the finality bisimulation condition demands that q_0 be final, which it is not. From this contradiction, we conclude that q_0 and s_0 are *not* bisimilar. □

However, if both automata are DFAs, then language equivalence does imply bisimilarity.

THEOREM 2.5. *Let $\mathcal{A} = (Q, \delta, F)$ be* a DFA *over an input alphabet Σ. Then $\mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)$ implies $q \sim s$.*

Proof. Let $\mathcal{R} = \{(q, s) \mid \mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)\}$; we will prove that $\mathcal{R}$ is a bisimulation on the DFA $\mathcal{A}$. As the largest bisimulation, bisimilarity will then contain $\mathcal{R}$.

Input bisimulation Assume that $q \mathrel{\mathcal{R}} s \xrightarrow{a} s'_a$ for some state s; we must show that $q \xrightarrow{a} \mathcal{R}\, s'_a$. Because $\mathcal{A}$ is deterministic, it suffices to show that $\mathcal{L}_{\mathcal{A}}(q'_a) = \mathcal{L}_{\mathcal{A}}(s'_a)$, where q'_a is the unique a-successor of q.

Choose an arbitrary word w from the language accepted from state q'_a. Then $a\,w$ is in the language accepted from state q, and, because q and s are $\mathcal{R}$-related, also in the language accepted from s. Because $\mathcal{A}$ is deterministic, this can only be if w is in the language accepted from state s'_a, the unique a-successor of s. Thus $\mathcal{L}_{\mathcal{A}}(q'_a) \subseteq \mathcal{L}_{\mathcal{A}}(s'_a)$. By symmetric reasoning, $\mathcal{L}_{\mathcal{A}}(q'_a) \supseteq \mathcal{L}_{\mathcal{A}}(s'_a)$. It follows that $q'_a \mathrel{\mathcal{R}} s'_a$.

Finality bisimulation Assume that $q \mathrel{\mathcal{R}} s \in F$; we must show that $q \in F$. Because s is a final state, it accepts the empty word, ϵ. The state q must also accept the empty word, because $\mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)$. A state accepts the empty word only if it is a final state, so it follows that $q \in F$. □

2.4 *Infinite-word sequential transducers*

Sequential and subsequential transducers[4] are usually described in terms of finite words. In this document, we are interested in transducers for their computational behavior rather than for the functions they induce. This, together with technical considerations that will become apparent in chapter 9, means that we are only interested in *infinite*-word sequential transducers.

[4] Ginsburg and Rose 1966; Schützenberger 1977.

The following definition is adapted from Béal and Carton.[5]

[5] Béal and Carton 2002.

DEFINITION 2.4. An *infinite-word sequential transducer* over the finite input and output alphabets Σ and Γ, respectively, is a triple $\mathcal{T} = (Q, \delta, \sigma)$ consisting of:

- a finite set of *states*, Q;
- a *transition function*, $\delta\colon Q \times \Sigma \to Q$; and
- an *output function*, $\sigma\colon Q \times \Sigma \to \Gamma^*$.

We may define a function $\Downarrow\colon Q \times \Sigma^\omega \to \Gamma^\omega$ as the largest function such that $(q, a\,w) \Downarrow \sigma(q,a)\,v$ if $(\delta(q,a), w) \Downarrow v$. The transducer $\mathcal{T}$ maps, from state q, the infinite input word w into the infinite output word v if $(q, w) \Downarrow v$.

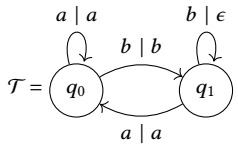


Figure 2.4: An infinite-word sequential transducer that compresses runs of consecutive bs

Example 2.3. As a concrete example consider the infinite-word transducer $\mathcal{T}$ over the input and output alphabets $\Sigma = \Gamma = \{a, b\}$ that is depicted in fig. 2.4. This transducer maps, from state q_0, infinite input words into words in which each run of bs has been compressed into a single b. For instance, it maps $w = abbaabbba\cdots$ to $v = abaaba\cdots$ because $(q_0, w) \Downarrow v$. □

2.5 *Turing machines*

In section 9.3.3, we will construct session-typed processes that represent Turing machines. We are interested in how these machines compute, but not interested in what functions they compute. In other words, our focus is on computation, not computability.

This means that our definitions of Turing machines differ from the definitions traditionally used, such as in Hopcroft et al.'s text.[6] For example, we use infinite words to describe truly infinite tapes, rather than using finite words to describe the frontiers of unbounded tapes, and our machines also have no notion of acceptance. The specific reasons for these differences will become clear in section 9.3.3.

[6] Hopcroft et al. 2006.

Definition 2.5. A *two-way infinite tape Turing machine* over a finite alphabet Σ is a pair $\mathcal{M} = (Q, \delta)$ consisting of:

- a finite set of *states*, Q; and
- a *transition function* $\delta\colon Q \times \Sigma \to Q \times \Sigma \times \{\mathsf{L}, \mathsf{R}\}$.

The two-way infinite tape is divided into two one-way halves, with the machine's *head* placed between the two halves. A *configuration* of $\mathcal{M}$ is thus either $w \triangleleft q \,\square\, v$ or $w \,\square\, q \triangleright v$ for left-infinite word $w \in \Sigma^{\bar{\omega}}$, right-infinite word $v \in \Sigma^\omega$, and state $q \in Q$.[7] The machine's head faces either the left- or right-hand half of the tape, as indicated by the notation surrounding the state q.

[7] In other words, configurations are drawn from $\Sigma^{\bar{\omega}} \times (\bigcup_{q\in Q} \{\triangleleft\, q \,\square\,,\ \square\, q \triangleright\}) \times \Sigma^\omega$.

To describe the machine's moves, we define a function $\longrightarrow$ on configurations. For a left-facing head, this function is given by:

$$w\,a \triangleleft q \,\square\, v \longrightarrow \begin{cases} w \triangleleft q' \,\square\, b\,v & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ w\,b \,\square\, q' \triangleright v & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases}$$

Symmetrically, for a right-facing head, this function is given by:

$$w \,\square\, q \triangleright a\,v \longrightarrow \begin{cases} w \triangleleft q' \,\square\, b\,v & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ w\,b \,\square\, q' \triangleright v & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases}$$

The direction that the head faces indicates the next symbol to be read from the tape. When a left-facing head is instructed to move right or a right-facing head is instructed to move left, the head's direction changes but its placement between the two tape halves does not.

Definition 2.6. A *one-way infinite tape Turing machine* over a finite alphabet Σ is a tuple $\mathcal{M} = (Q, \delta, F)$ consisting of:

- a finite set of *states*, Q;
- a *transition function* $\delta\colon Q \times \Sigma \to Q \times \Sigma \times \{\mathsf{L}, \mathsf{R}\}$; and
- a subset of *final states*, $F \subseteq Q$.

Because the tape is only one-way infinite, a configuration of machine $\mathcal{M}$ is either $w \triangleleft q \square v$ or $w \square q \triangleright v$ for left-infinite word $w \in \Sigma^{\bar{\omega}}$, *finite* word $v \in \Sigma^*$, and state $q \in Q$; *i.e.*, configurations are drawn from $\Sigma^{\bar{\omega}} \times (\bigcup_{q \in Q} \{\triangleleft q \square, \square q \triangleright\}) \times \Sigma^*$.

To describe the machine's moves, we define a function $\longrightarrow$ on configurations. For a left-facing head, this function is given by:

$$w\, a \triangleleft q \square v \longrightarrow \begin{cases} w \triangleleft q' \square b\, v & \text{if } \delta(q, a) = (q', b, \mathsf{L}) \\ w\, b \square q' \triangleright v & \text{if } \delta(q, a) = (q', b, \mathsf{R}) \end{cases}$$

This is exactly as it was for the two-way infinite tape Turing machines. Because the tape is only one-way infinite, the right-facing head's treatment differs:

$$w \square q \triangleright a v \longrightarrow \begin{cases} w \triangleleft q' \square b\, v & \text{if } \delta(q, a) = (q', b, \mathsf{L}) \\ w\, b \square q' \triangleright v & \text{if } \delta(q, a) = (q', b, \mathsf{R}) \end{cases}$$

$$w \square q \triangleright \epsilon \longrightarrow \begin{cases} w & \text{if } q \in F \\ w \triangleleft q \square \epsilon & \text{if } q \notin F \end{cases}$$

When a right-facing head reaches the finite end of the tape, its behavior depends on whether the state is final. If the state is final, the machine terminates; otherwise, the machine remains in the same state but effectively moves left one symbol (by turning to face left).

Notice that the machine does not terminate as soon as it enters a final state – it must also reach the finite end of the tape. It is up to the machine's programmer, by appropriately crafting the transition function δ, to ensure that final states eventually lead the machine to the tape's finite end.

3
Ordered logic

In its traditional form, intuitionistic logic[1] presumes that hypotheses admit three structural properties: *weakening*, that hypotheses need not be used; *contraction*, that hypotheses may be reused indefinitely; *exchange*, that hypotheses may be freely permuted.

[1] And classical logic, too.

Substructural logics are so named because they reject some or all of these structural properties. Most famously, linear logic[2] is substructural because it rejects both weakening and contraction. The result is a system in which each hypothesis must be used exactly once; accordingly, linear hypotheses may be viewed as consumable resources.[3]

[2] Girard 1987.

[3] Girard 1987.

Ordered logic, also known as the (full) Lambek calculus,[4] goes a substructural step further. Like its linear cousin, ordered logic rejects weakening and contraction, making ordered hypotheses resources, too. But ordered logic additionally eschews exchange; ordered hypotheses are resources that must remain in order, with no reshuffling.

[4] Lambek 1958, 1961; Abrusci 1990; Kanazawa 1992.

This chapter serves to review a sequent calculus presentation of ordered logic. Lambek leveraged the noncommutativity of antecedents to give a formal description of sentence structure. In this work, however, our interest is not mathematical linguistics but the logical foundations of concurrent computation. Accordingly, the description of ordered logic in this chapter has a proof-theoretic emphasis and is derived from presentations by Polakow and Pfenning.[5]

[5] Polakow and Pfenning 1999b; Pfenning 2016.

Section 3.1 introduces ordered logic's sequent calculus as a collection of inference rules, informally justifying them with a resource interpretation similar to that of linear logic.

For this collection of rules to constitute a well-defined logic, it must have a verificationist meaning-theory in the tradition of Gentzen, Dummett, and Martin-Löf.[6] Together, the cut elimination and identity elimination theorems (theorems 3.2 and 3.4, section 3.2) serve to establish a proof normalization result: every proof has a corresponding verification.

[6] Gentzen 1935; Dummett 1976; Martin-Löf 1983.

Section 3.3 sketches an extension of the ordered sequent calculus with circular propositions and proofs, and section 3.4 briefly describes several other

extensions that are possible.

The reader who is familiar with ordered logic's sequent calculus and its basic metatheory – particularly the cut elimination result – should feel free to skip this chapter.

3.1 *A sequent calculus presentation of ordered logic*

The full sequent calculus for ordered logic will be summarized in fig. 3.2, but first we will discuss the calculus's judgmental principles and logical connectives one by one.

3.1.1 *Sequents and contexts*

SEQUENTS In ordered logic's sequent calculus presentation, the basic judgment is a sequent

$$A_1\, A_2 \cdots A_n \vdash A\,,$$

where the propositions $A_1, A_2, \ldots, A_n$ are assumptions, or *antecedents*, that are arranged into an ordered list; the proposition A is termed the *consequent*.

Ordered logic eschews the usual structural properties of antecedents – namely weakening, contraction, and exchange. As in linear logic, the absence of weakening and contraction means that antecedents may neither be discarded nor duplicated within a proof. Neither a proof of $A_2 \cdots A_n \vdash A$ nor of $A_1\, A_1\, A_2 \cdots A_n \vdash A$ implies a proof of $A_1\, A_2 \cdots A_n \vdash A$, for example. But ordered logic's rejection of the exchange property takes things one step further: antecedents may not even be permuted within a proof. For example, $A_2\, A_1 \cdots A_n \vdash A$ does not imply $A_1\, A_2 \cdots A_n \vdash A$.

Like linear sequents,[7] ordered sequents can be given a resource interpretation – but with a slight twist. A proof of an ordered sequent $A_1\, A_2 \cdots A_n \vdash A$ can be interpreted as a recipe for producing resource A from the resources $A_1\, A_2 \cdots A_n$. The small twist is that these resources are inherently ordered and may not be permuted, exactly because ordered logic rejects the exchange property that linear logic admits.

[7] Girard 1987.

CONTEXTS To keep the notation for sequents concise, the list of antecedents is usually packaged into an ordered context $\Omega = A_1\, A_2 \cdots A_n$, with the sequent then written $\Omega \vdash A$.[8] Algebraically, ordered contexts Ω form a free noncommutative monoid:

$$\Omega, \Delta ::= \Omega_1\, \Omega_2 \mid \cdot \mid A\,,$$

where the monoid operation is concatenation, denoted by juxtaposition, and the unit element is the empty context, denoted by $(\cdot)$. (We will also sometimes use the metavariable Δ for *ordered* contexts.) As a monoid, ordered contexts are equivalent up to associativity and unit laws (see adjacent figure).

[8] We will also sometimes use Δ as an additional metavariable for ordered contexts.

$$(\Omega_1\, \Omega_2)\, \Omega_3 = \Omega_1\, (\Omega_2\, \Omega_3)$$
$$(\cdot)\, \Omega = \Omega = \Omega\, (\cdot)$$

Figure 3.1: Monoid laws for ordered contexts

We choose to keep this equivalence implicit, however, treating equivalent contexts as syntactically indistinguishable.[9] Associativity means that contexts are indeed lists, not trees; and noncommutativity means that those lists are ordered, not multisets as in linear logic.

[9] Throughout this document, we will encounter free noncommutative monoids in various guises. Each time, we will choose to keep the equivalence induced by the monoid laws implicit, as we do here.

3.1.2 *Judgmental principles*

Even without considering the specific structure of propositions, two judgmental principles must hold if sequents are to accurately describe the production of resources.

First, given a resource A, producing the same resource A should be effortless – it already exists! This amounts to an identity principle for sequents:

Identity principle $A \vdash A$ for all propositions A.

This principle is adopted by the ordered sequent calculus as a primitive rule of inference:

$$\frac{}{A \vdash A}\ \textsc{id}^A\ .$$

Both the identity principle and its corresponding ID rule capture the idea that resource production is a reflexive process.

Second, and dually, resource production should be transitive process. If a resource B can be produced from resource A (*i.e.*, $A \vdash B$), and if a resource C can be produced from resource B (*i.e.*, $B \vdash C$), then, by chaining the productions, it ought to be possible to produce C from A (*i.e.*, $A \vdash C$). For sequents, this amounts to a cut principle that is most useful in a generalized form:

Cut principle If $\Omega \vdash B$ and $\Omega'_L\, B\, \Omega'_R \vdash C$, then $\Omega'_L\, \Omega\, \Omega'_R \vdash C$.

As with the identity principle, this cut principle is adopted by the ordered sequent calculus as a primitive rule of inference:

$$\frac{\Omega \vdash B \qquad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^B\ .$$

The importance of these two judgmental principles goes beyond that of mere rules of inference. As we will see in section 3.2, the admissibility of these principles serves an important role in defining the meaning of the logical connectives.

3.1.3 *The ordered logical connectives*

The propositions of ordered logic are given by the following grammar.

$$\textsc{propositions} \quad A, B, C ::= a \mid A \bullet B \mid \mathbf{1} \mid A \circ B \mid A \oplus B \mid \mathbf{0} \mid A \mathbin{\&} B \mid \top \mid A \setminus B \mid B \mathbin{/} A$$

Among these are propositional atoms, a, which stand in for arbitrary propositions. The other propositions are built up from these atoms by using the logical connectives.

Under the resource interpretation of ordered logic, these logical connectives may be viewed as resource constructors. A connective's right rule defines how to produce that kind of resource, while the corresponding left rules define how that kind of resource may be used.

ORDERED CONJUNCTION AND ITS UNIT Ordered conjunction[10] is the proposition $A \bullet B$, read "A fuse B". Under the resource interpretation, $A \bullet B$ is the side-by-side pair of resources A and B, packaged as a single ordered resource. Its sequent calculus inference rules are:

$$\frac{\Omega_1 \vdash A \quad \Omega_2 \vdash B}{\Omega_1\ \Omega_2 \vdash A \bullet B}\ \bullet\text{R} \qquad \frac{\Omega'_L\ A\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash C}\ \bullet\text{L}$$

The right rule, $\bullet$R, says that $A \bullet B$ may be produced by partitioning the available resources into $\Omega_1\ \Omega_2$ and then separately using the resources Ω_1 and Ω_2 to produce A and B, respectively. The left rule, $\bullet$L, shows how to use resource $A \bullet B$: simply unwrap the package to leave the separate contents, resources A and B, side by side.

Just as truth is the nullary analogue of conjunction in intuitionistic logic, multiplicative truth, $\mathbf{1}$, is the nullary analogue of ordered conjunction. Under the resource interpretation, $\mathbf{1}$ is therefore an empty resource package that contains no resources.

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}\text{R} \qquad \frac{\Omega'_L\ \Omega'_R \vdash C}{\Omega'_L\ \mathbf{1}\ \Omega'_R \vdash C}\ \mathbf{1}\text{L}$$

The sequents $\mathbf{1} \bullet A \dashv\vdash A \dashv\vdash A \bullet \mathbf{1}$ are all derivable[11], so $\mathbf{1}$ is indeed $\bullet$'s unit.

In addition to $A \bullet B$, the proposition $A \circ B$, read "A twist B", is included. Under the resource interpretation, $A \circ B$ is the side-by-side pair of resources B and A, packaged as a single ordered resource. If we gave sequent calculus inference rules for $A \circ B$, the sequents $B \bullet A \dashv\vdash A \circ B$ and $\mathbf{1} \circ A \dashv\vdash A \dashv\vdash A \circ \mathbf{1}$ would all be derivable. Therefore, instead of taking $A \circ B$ as a primitive and explicitly giving it inference rules, we choose to treat it as merely a notational definition for the ordered conjunction $B \bullet A$.

DISJUNCTION AND ITS UNIT Disjunction is the proposition $A \oplus B$, read "A plus B".[12] Under the resource interpretation, $A \oplus B$ is a package that contains one of the resources A or B (but not both).

$$\frac{\Omega \vdash A}{\Omega \vdash A \oplus B}\ \oplus\text{R}_1 \qquad \frac{\Omega \vdash B}{\Omega \vdash A \oplus B}\ \oplus\text{R}_2 \qquad \frac{\Omega'_L\ A\ \Omega'_R \vdash C \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \oplus B)\ \Omega'_R \vdash C}\ \oplus\text{L}$$

The right rules, $\oplus$R$_1$ and $\oplus$R$_2$, say that a resource $A \oplus B$ may be produced from the resources Ω by producing either A or B and then wrapping that resource up as an $A \oplus B$ package. The left rule, $\oplus$L, shows how to use a resource $A \oplus B$: unwrap the package and use whatever it contains – whether an A or a B.

[10] Also known as multiplicative conjunction.

[11] $A \dashv\vdash B$ stands for the sequents $A \vdash B$ and $B \vdash A$.

[12] This connective is also known as additive disjunction, in contrast with the multiplicative disjunction of classical linear logic; being intuitionistic, ordered logic does not have a purely multiplicative disjunction. See Chang et al. (2003).

Falsehood, $\mathbf{0}$, can be viewed as the nullary analogue of disjunction:

$$\text{(no } \mathbf{0}\text{R rule)} \qquad \frac{}{\Omega'_L\ \mathbf{0}\ \Omega'_R \vdash C}\ \mathbf{0}\text{L}$$

The sequents $\mathbf{0} \oplus A \dashv\vdash A \dashv\vdash A \oplus \mathbf{0}$ are all derivable, so $\mathbf{0}$ is indeed $\oplus$'s unit.

ALTERNATIVE CONJUNCTION AND ITS UNIT Alternative conjunction[13] is the proposition $A \,\&\, B$, read "A with B"; it is dual to disjunction. Under the resource interpretation, $A \,\&\, B$ is the resource that can be transformed – irreversibly – into either a resource A or a resource B, whichever the user chooses.

[13] Also known as additive conjunction.

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \,\&\, B}\ \&\text{R} \qquad \frac{\Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ (A \,\&\, B)\ \Omega'_R \vdash C}\ \&\text{L}_1 \qquad \frac{\Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \,\&\, B)\ \Omega'_R \vdash C}\ \&\text{L}_2$$

The left rules, $\&\text{L}_1$ and $\&\text{L}_2$, show how to use a resource $A \,\&\, B$: transform it into either an A or a B and then use that resource. The right rule, $\&$R, says that to produce a resource $A \,\&\, B$ the producer must be prepared to produce either A or B – whichever the user eventually chooses.

Additive truth, $\top$, can be viewed as the nullary analogue of alternative conjunction:

$$\frac{}{\Omega \vdash \top}\ \top\text{R} \qquad \text{(no } \top\text{L rule)}$$

Once again, the sequents $\top \,\&\, A \dashv\vdash A \dashv\vdash A \,\&\, \top$ are all derivable, so $\top$ is indeed the unit of $\&$.

LEFT- AND RIGHT-HANDED IMPLICATIONS Left-handed implication is the proposition $A \setminus B$, read "A under B" or "A left-implies B". When interpreted as a resource, $A \setminus B$ is the resource that can transform a left-adjacent resource A into the resource B.

$$\frac{A\ \Omega \vdash B}{\Omega \vdash A \setminus B}\ \setminus\text{R} \qquad \frac{\Omega \vdash A \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ \Omega\ (A \setminus B)\ \Omega'_R \vdash C}\ \setminus\text{L}$$

The left rule, $\setminus$L, shows how to use a resource $A \setminus B$: first produce A from the left-adjacent resources Ω, then transform the left-adjacent A into the resource B, and finally use that B. The right rule, $\setminus$R, says that resources Ω can produce $A \setminus B$ if the same resources prefixed with A – that is, $A\ \Omega$ – can produce B.

Right-handed implication, $B \,/\, A$ (read "B over A" or "A right-implies B"), is symmetric to left-handed implication: $B \,/\, A$ is the resource that can transform a *right*-adjacent resource A into the resource B.

$$\frac{\Omega\ A \vdash B}{\Omega \vdash B \,/\, A}\ /\text{R} \qquad \frac{\Omega \vdash A \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (B \,/\, A)\ \Omega\ \Omega'_R \vdash C}\ /\text{L}$$

The two forms of implication each enjoy their own currying laws: the sequents $A \setminus (B \setminus C) \dashv\vdash (A \circ B) \setminus C$ and $(C \,/\, B) \,/\, A \dashv\vdash C \,/\, (A \bullet B)$ are derivable.

Ordered logical connective		*Resource interpretation*
Ordered conjunction	$A \bullet B$	A side-by-side pair of resources A and B
Multiplicative truth	$\mathbf{1}$	The unit of ordered conjunction
Twisted conjunction	$A \circ B$	A side-by-side pair of resources B and A
(Additive) disjunction	$A \oplus B$	A package containing A or B (not both)
(Additive) falsehood	$\mathbf{0}$	A package containing no resources
Alternative conjunction	$A \mathbin{\&} B$	A resource that transforms into A or B
Additive truth	$\top$	An immutable resource
Left-handed implication	$A \setminus B$	Transforms a left-adjacent A into B
Right-handed implication	$B \mathbin{/} A$	Transforms a right-adjacent A into B

Table 3.1: A resource interpretation of the ordered logical connectives

PROPOSITIONS $A, B, C ::= a \mid A \bullet B \mid \mathbf{1} \mid A \circ B \mid A \oplus B \mid \mathbf{0} \mid A \mathbin{\&} B \mid \top \mid A \setminus B \mid B \mathbin{/} A$

CONTEXTS $\Omega, \Delta ::= \Omega_1\ \Omega_2 \mid \cdot \mid A$

$$\frac{\Omega \vdash A \quad \Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ \Omega\ \Omega'_R \vdash C}\ \textsc{cut}^A \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{\Omega_1 \vdash A \quad \Omega_2 \vdash B}{\Omega_1\ \Omega_2 \vdash A \bullet B}\ \bullet\textsc{r} \qquad \frac{\Omega'_L\ A\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash C}\ \bullet\textsc{l}$$

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}\textsc{r} \qquad \frac{\Omega'_L\ \Omega'_R \vdash C}{\Omega'_L\ \mathbf{1}\ \Omega'_R \vdash C}\ \mathbf{1}\textsc{l}$$

$$A \circ B \stackrel{\text{def}}{=} B \bullet A$$

$$\frac{\Omega \vdash A}{\Omega \vdash A \oplus B}\ \oplus\textsc{r}_1 \qquad \frac{\Omega \vdash B}{\Omega \vdash A \oplus B}\ \oplus\textsc{r}_2 \qquad \frac{\Omega'_L\ A\ \Omega'_R \vdash C \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \oplus B)\ \Omega'_R \vdash C}\ \oplus\textsc{l}$$

(no $\mathbf{0}\textsc{r}$ rule) $\qquad \frac{}{\Omega'_L\ \mathbf{0}\ \Omega'_R \vdash C}\ \mathbf{0}\textsc{l}$

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \mathbin{\&} B}\ \&\textsc{r} \qquad \frac{\Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ (A \mathbin{\&} B)\ \Omega'_R \vdash C}\ \&\textsc{l}_1 \qquad \frac{\Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \mathbin{\&} B)\ \Omega'_R \vdash C}\ \&\textsc{l}_2$$

$$\frac{}{\Omega \vdash \top}\ \top\textsc{r} \qquad \text{(no } \top\textsc{l} \text{ rule)}$$

$$\frac{A\ \Omega \vdash B}{\Omega \vdash A \setminus B}\ \setminus\textsc{r} \qquad \frac{\Omega \vdash A \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ \Omega\ (A \setminus B)\ \Omega'_R \vdash C}\ \setminus\textsc{l}$$

$$\frac{\Omega\ A \vdash B}{\Omega \vdash B \mathbin{/} A}\ /\textsc{r} \qquad \frac{\Omega \vdash A \quad \Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (B \mathbin{/} A)\ \Omega\ \Omega'_R \vdash C}\ /\textsc{l}$$

Figure 3.2: A summary of ordered logic's sequent calculus, as presented in section 3.1

3.2 *A verificationist meaning-theory of the ordered sequent calculus*

The previous section presented a collection of inference rules that have an apparently sensible resource interpretation. But how can we be sure that the rules constitute a well-defined *logic*?

In the tradition of Gentzen, Dummett, and Martin-Löf, a logic is well-defined if it rests on the solid foundation of a verificationist meaning-theory.[14] In Martin-Löf's words, "The meaning of a proposition is determined by [...] what counts as a verification of it." And a verification is a proof that decomposes that proposition into its subformulas, without dragging in other, unrelated propositions. In this way, the meaning of a proposition is compositional.

[14] Gentzen 1935; Dummett 1976; Martin-Löf 1983.

For the ordered sequent calculus, a verification is thus a proof that relies only on the right and left inference rules (and the $\textsc{id}^a$ rule for propositional atoms) – the CUT rule drags in an unrelated proposition as its cut formula; and, when A is a compound proposition, the $\textsc{id}^A$ rule fails to decompose A to its subformulas. A proof is *cut-free* if it does not contain any instances of the CUT rule; similarly, a proof is *identity-long* if all instances of the ID rule occur at propositional atoms. Verifications are thus exactly those proofs that are both cut-free and identity-long.

Because meaning is based on verifications, every proof must have a corresponding verification if proofs are to be meaningful. That is, we need to describe a procedure for normalizing arbitrary proofs to verifications. The characterization of verifications as cut-free, identity-long proofs suggests a two-step strategy for proof normalization:

1. Eliminate all instances of CUT.
2. Without introducing new instances of CUT, eliminate all remaining instances of ID that occur at non-atomic propositions.

The end result will be a cut-free, identity-long proof – a verification.

This normalization procedure is described by the constructive content of the following theorems; their proofs amount to defining functions on proofs.

THEOREM 3.2 (Cut elimination). *If a proof of* $\Omega \vdash A$ *exists, then there exists a* cut-free *proof of* $\Omega \vdash A$.

THEOREM 3.4 (Identity elimination). *If a proof of* $\Omega \vdash A$ *exists, then an* identity-long *proof of* $\Omega \vdash A$ *exists. Moreover, if the given proof is cut-free, so is the identity-long proof.*

COROLLARY 3.5 (Proof normalization). *If a proof of* $\Omega \vdash A$ *exists, then a verification (i.e., a cut-free, identity-long proof) of* $\Omega \vdash A$ *exists.*

We will now prove these theorems.

3.2.1 *Cut elimination*

To prove the cut elimination theorem stated above, we will eventually use a straightforward induction on the structure of the given proof. But first, we

need to establish a cut principle for cut-free proofs:

Lemma 3.1 (Admissibility of cut). *If cut-free proofs of* $\Omega \vdash A$ *and* $\Omega'_L\, A\, \Omega'_R \vdash C$ *exist, then there exists a* cut-free *proof of* $\Omega'_L\, \Omega\, \Omega'_R \vdash C$.

Before proceeding to this lemma's proof, it is worth emphasizing a subtle distinction between the sequent calculus's primitive CUT rule and the admissible cut principle that this lemma establishes. To be completely formal, we ought to treat cut-freeness as an extrinsic, Curry-style property of proofs and indicate that property by decorating the turnstile: a proof of $\Omega \vdash^{\text{cf}} A$ is a cut-free proof of $\Omega \vdash A$. The admissible cut principle stated in lemma 3.1 could then be expressed as the rule

$$\frac{\Omega \vdash^{\text{cf}} A \qquad \Omega'_L\, A\, \Omega'_R \vdash^{\text{cf}} C}{\Omega'_L\, \Omega\, \Omega'_R \vdash^{\text{cf}} C}\ \text{A-CUT}^A$$

with the dotted line indicating that this is an admissible, not primitive, rule. Writing it in this way emphasizes that the proof of lemma 3.1 will amount to defining a meta-level function that takes cut-free proofs of $\Omega \vdash A$ and $\Omega'_L\, A\, \Omega'_R \vdash C$ and produces a *cut-free* proof of $\Omega'_L\, \Omega\, \Omega'_R \vdash C$. Contrast this with the primitive CUT rule of the ordered sequent calculus, which forms a (cut-full) proof of $\Omega'_L\, \Omega\, \Omega'_R \vdash C$ from (potentially cut-full) proofs of $\Omega \vdash A$ and $\Omega'_L\, A\, \Omega'_R \vdash C$.

From here on, we won't bother to be quite so pedantic, instead often omitting the turnstile decoration on cut-free proofs, with the understanding that any proofs to which the admissible A-CUT rule is applied are necessarily cut-free.[15]

[15] The distinction will become somewhat more important in chapter 8 when we introduce a "semi-axiomatic sequent calculus" for singleton logic.

With that clarification out of the way, we may proceed to proving the previously stated lemma and theorem.

Lemma 3.1 (Admissibility of cut). *If cut-free proofs of* $\Omega \vdash A$ *and* $\Omega'_L\, A\, \Omega'_R \vdash C$ *exist, then there exists a* cut-free *proof of* $\Omega'_L\, \Omega\, \Omega'_R \vdash C$.

Proof. This lemma was proved in a similar setting by Polakow and Pfenning[16] using a standard technique for proving the admissibility of a cut principle[17] – a lexicographic structural induction, first on the structure of the cut formula, A, and then on the structures of the given proofs. We review their proof here.

[16] Polakow and Pfenning 1999b.

[17] Pfenning 1995.

As usual, the various cases can be classified into three categories: identity cases, principal cases, and commutative cases.

Identity cases In the cases where one of the two proofs is an instance of the ID rule, the admissible cut can be reduced to the other proof alone. For example:

$$\frac{\dfrac{}{A \vdash A}\ \text{ID}^A \qquad \overset{\mathcal{E}}{\Omega'_L\, A\, \Omega'_R \vdash C}}{\Omega'_L\, A\, \Omega'_R \vdash C}\ \text{A-CUT}^A \quad = \quad \overset{\mathcal{E}}{\Omega'_L\, A\, \Omega'_R \vdash C}$$

That the cut and identity principles are inverses is reflected in these identity cases.

Principal cases In another class of cases, both proofs end by introducing the cut formula – on the right in the left-hand proof with a right rule, and on the left in the right-hand proof with a left rule. These cases are resolved by reducing the admissible cut to several instances of the admissible cut principle at proper subformulas of the cut formula.

For example, the principal cut reduction for $A_1 \setminus A_2$ yields cuts at the proper subformulas A_1 and A_2:

$$\dfrac{\dfrac{\overset{\mathcal{D}_1}{A_1\ \Omega \vdash A_2}}{\Omega \vdash A_1 \setminus A_2}\ \setminus\text{R} \qquad \dfrac{\overset{\mathcal{E}_1}{\Omega_1' \vdash A_1} \qquad \overset{\mathcal{E}_2}{\Omega_L'\ A_2\ \Omega_R' \vdash C}}{\Omega_L'\ \Omega_1'\ (A_1 \setminus A_2)\ \Omega_R' \vdash C}\ \setminus\text{L}}{\Omega_L'\ \Omega_1'\ \Omega\ \Omega_R' \vdash C}\ \text{A-CUT}^{A_1 \setminus A_2}$$

$$=$$

$$\dfrac{\dfrac{\overset{\mathcal{D}_1}{\Omega_1' \vdash A_1} \qquad \overset{\mathcal{E}_1}{A_1\ \Omega \vdash A_2}}{\Omega_1'\ \Omega \vdash A_2}\ \text{A-CUT}^{A_1} \qquad \overset{\mathcal{E}_2}{\Omega_L'\ A_2\ \Omega_R' \vdash C}}{\Omega_L'\ \Omega_1'\ \Omega\ \Omega_R' \vdash C}\ \text{A-CUT}^{A_2}$$

Commutative cases In the remaining cases, at least one of the two proofs ends by introducing a side formula, *i.e.*, a formula other than the cut formula. To reduce the admissible cut, it is permuted with the final inference in that proof; the reduced instance of the admissible cut is smaller because it occurs with the same cut formula but smaller proofs.

Commutative cases are subcategorized as left- or right-commutative cut reductions according to the branch into which the admissible cut is permuted. For example, one right-commutative case involves a right-hand proof that ends by introducing the consequent with the $\setminus$R rule:

$$\dfrac{\overset{\mathcal{D}}{\Omega \vdash A} \qquad \dfrac{\overset{\mathcal{E}_1}{C_1\ \Omega_L'\ A\ \Omega_R' \vdash C_2}}{\Omega_L'\ A\ \Omega_R' \vdash C_1 \setminus C_2}\ \setminus\text{R}}{\Omega_L'\ \Omega\ \Omega_R' \vdash C_1 \setminus C_2}\ \text{A-CUT}^{A} \quad = \quad \dfrac{\dfrac{\overset{\mathcal{D}}{\Omega \vdash A} \qquad \overset{\mathcal{E}_1}{C_1\ \Omega_L'\ A\ \Omega_R' \vdash C_2}}{C_1\ \Omega_L'\ \Omega\ \Omega_R' \vdash C_2}\ \text{A-CUT}^{A}}{\Omega_L'\ \Omega\ \Omega_R' \vdash C_1 \setminus C_2}\ \setminus\text{R}$$

Among the other right-commutative cases are several involving a right-hand proof that ends by using a left rule, such as the $\setminus$L rule, to introduce a side formula. This contrasts with the left-commutative cases: the left-hand proof can never use a right rule to introduce a side formula because its only consequent is the cut formula. □

With the admissibility of a cut principle for cut-free proofs established, we may finally prove a cut elimination result.

THEOREM 3.2 (Cut elimination). *If a proof of* $\Omega \vdash A$ *exists, then there exists a* cut-free *proof of* $\Omega \vdash A$.

Proof. We follow the proof sketched by Polakow and Pfenning.[18] The proof is by structural induction on the proof of $\Omega \vdash A$, with appeals to the admissibility of cut (lemma 3.1) whenever a CUT rule is encountered.

[18] Polakow and Pfenning 1999b.

Like the admissibility of cut lemma, this theorem may be rendered as an admissible rule:

$$\dfrac{\Omega \vdash A}{\Omega \vdash^{\mathrm{cf}} A}\ \textsc{ce}$$

Writing the theorem in this way serves to emphasize that its proof amounts to the definition of a meta-level function for normalizing proofs to cut-free form.

The crucial case is then resolved as follows:

$$\dfrac{\dfrac{\Omega \vdash A \quad \Omega'_L\, A\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^A}{\Omega'_L\, \Omega\, \Omega'_R \vdash^{\mathrm{cf}} C}\ \textsc{ce} \quad = \quad \dfrac{\dfrac{\Omega \vdash A}{\Omega \vdash^{\mathrm{cf}} A}\ \textsc{ce} \quad \dfrac{\Omega'_L\, A\, \Omega'_R \vdash C}{\Omega'_L\, A\, \Omega'_R \vdash^{\mathrm{cf}} C}\ \textsc{ce}}{\Omega'_L\, \Omega\, \Omega'_R \vdash^{\mathrm{cf}} C}\ \textsc{a-cut}^A$$

All other cases are resolved compositionally. □

3.2.2 *Identity elimination*

By this cut elimination theorem, an arbitrary proof may be put into cut-free form. Recall from earlier in this section that the next step toward proof normalization is to eliminate all remaining instances of the ID rule that occur at non-atomic propositions A. Before proving identity elimination, we need to prove that an identity principle is admissible for identity-long proofs.

LEMMA 3.3 (Admissibility of identity). *For all propositions A, an* identity-long *proof of $A \vdash A$ exists. Moreover, this proof is cut-free.*

Proof. As usual, by induction on the structure of the proposition A. As before, we may represent this lemma as an admissible rule:

$$\dfrac{}{A \vdash^{\mathrm{il}} A}\ \textsc{a-id}^A$$

to suggest that this proof amounts to defining a meta-level function on propositions A.

In the base case of propositional atoms a, the instance of the ID rule at a is itself already identity-long:

$$\dfrac{}{a \vdash^{\mathrm{il}} a}\ \textsc{a-id}^a \quad = \quad \dfrac{}{a \vdash^{\mathrm{il}} a}\ \textsc{id}^a$$

For non-atomic propositions, the identity-long proof of $A \vdash A$ is constructed from right and left rules, together with calls to the admissible A-ID rule at subformulas of A. For example, the identity expansion at $A_1 \setminus A_2$ is:

$$\dfrac{}{A_1 \setminus A_2 \vdash^{\mathrm{il}} A_1 \setminus A_2}\ \textsc{a-id}^{A_1 \setminus A_2} \quad = \quad \dfrac{\dfrac{\dfrac{}{A_1 \vdash^{\mathrm{il}} A_1}\ \textsc{a-id}^{A_1} \quad \dfrac{}{A_2 \vdash^{\mathrm{il}} A_2}\ \textsc{a-id}^{A_2}}{A_1\ (A_1 \setminus A_2) \vdash^{\mathrm{il}} A_2}\ \setminus\textsc{l}}{A_1 \setminus A_2 \vdash^{\mathrm{il}} A_1 \setminus A_2}\ \setminus\textsc{r}$$

The remaining cases are similarly compositional. □

THEOREM 3.4 (Identity elimination). *If a proof of* $\Omega \vdash A$ *exists, then an* identity-long *proof of* $\Omega \vdash A$ *exists. Moreover, if the given proof is cut-free, so is the identity-long proof.*

Proof. As usual, by structural induction on the proof of $\Omega \vdash A$. Once again, we may represent this theorem as an admissible rule:

$$\dfrac{\Omega \vdash A}{\Omega \vdash^{\mathrm{il}} A}\ \textsc{ie}$$

The crucial case in the definition of this admissible rule comes when the given proof is instance of the ID rule. An appeal to the admissible A-ID rule (lemma 3.3) then yields an identity-long proof of $\Omega \vdash A$:

$$\dfrac{\dfrac{}{A \vdash A}\ \textsc{id}^A}{A \vdash^{\mathrm{il}} A}\ \textsc{ie} \quad = \dfrac{\vdots}{A \vdash^{\mathrm{il}} A}\ \textsc{a-id}^A$$

As part of lemma 3.3, we know that this proof is also cut-free.

The remaining cases are resolved compositionally. For example:

$$\dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ A_1\ \Omega \vdash A_2\end{array}}{\Omega \vdash A_1 \setminus A_2}\ \setminus\textsc{r}}{\Omega \vdash^{\mathrm{il}} A_1 \setminus A_2}\ \textsc{ie} \quad = \dfrac{\dfrac{\begin{array}{c}\mathcal{D}_1\\ A_1\ \Omega \vdash A_2\end{array}}{A_1\ \Omega \vdash^{\mathrm{il}} A_2}\ \textsc{ie}}{\Omega \vdash^{\mathrm{il}} A_1 \setminus A_2}\ \setminus\textsc{r}$$

Notice that no case introduces any instances of the CUT beyond those that were already present in the given proof. Thus, identity elimination preserves cut-freeness. □

3.2.3 *Proof normalization*

With the cut and identity elimination results (theorems 3.2 and 3.4) in hand, normalization of proofs to verification is a straightforward corollary:

COROLLARY 3.5 (Proof normalization). *If a proof of* $\Omega \vdash A$ *exists, then a verification (i.e., a cut-free, identity-long proof) of* $\Omega \vdash A$ *exists.*

Proof. Given a proof of $\Omega \vdash A$, applying cut elimination (theorem 3.2) and identity elimination (theorem 3.4) in sequence yields a proof that is both cut-free and long – in other words, a verification $\Omega \vdash^{\mathrm{cf,il}} A$. Using an admissible rule, this corollary may be represented as

$$\dfrac{\begin{array}{c}\mathcal{D}\\ \Omega \vdash A\end{array}}{\Omega \vdash^{\mathrm{cf,il}} A}\ \textsc{norm} \quad = \dfrac{\dfrac{\begin{array}{c}\mathcal{D}\\ \Omega \vdash A\end{array}}{\Omega \vdash^{\mathrm{cf}} A}\ \textsc{ce}}{\Omega \vdash^{\mathrm{cf,il}} A}\ \textsc{ie}$$

□

By establishing that every proof has a corresponding verification, we are now assured that the ordered sequent calculus presented in fig. 3.2 indeed constitutes a well-defined logic with a verificationist meaning-theory.

3.3 *Circular propositions and proofs*

By rejecting weakening and especially contraction, ordered logic as formulated above is bounded: there is exactly one of each antecedent, with no means of producing unbounded resources. The antecedent $A \setminus A \bullet A$ will indeed transform a left-adjacent resource A into a pair of resources, $A\,A$, effectively copying the initial A. But because the antecedent $A \setminus A \bullet A$ is itself a use-once resource, that is not enough to produce unbounded copies of A.

Linear logic traditionally introduces unboundedness with its 'of course' exponential, $!A$. The proposition $!A$ is viewed as an unbounded number of copies of resource A – as many, or as few, copies of A as desired.[19] Ordered logic can be extended with a related persistence modality,[20] so that $!A$ is an unbounded number of resources A, and similarly $!(A \setminus B)$ is a means for transforming, an unbounded number of times, a left-adjacent A into a B. As the notation suggests, the modality $!A$ is not unrelated to process replication, $!P$, in the π-calculus.[21] A replication $!P$ represents an unbounded number of processes P in parallel composition, $P \mid P \mid \cdots$, in much the same way as $!A$ represents an unbounded number of resources A.

[19] Girard 1987.

[20] Abrusci 1990; Polakow and Pfenning 1999a,b.

[21] Milner 1999.

But the modality $!A$ is not the only way to introduce unbounded behavior in ordered logic. As in the π-calculus,[22] recursive definitions are another path to unboundedness. Recursive definitions have been studied extensively,[23] but have not, to the best of our knowledge, been previously used in the context of ordered logic. We conjecture that persistence is strictly more expressive than recursive definitions

[22] Milner, Parrow, et al. 1992a,b.

[23] Hallnäs 1991; Eriksson 1991; Schroeder-Heister 1993; McDowell and Miller 2000; Tiu and Momigliano 2012.

In ordered logic, the difference between the exponential and recursive definitions is essentially one of mobility. The persistent resource $p \setminus A$ that derives from $!(p \setminus A)$ can be copied *anywhere* into the ordered context. By making the exponential a first-class logical connective, this mobility can be exploited. For example, the proposition $!(p \setminus a \bullet !p)$ would allow p to move anywhere within the ordered context. In contrast, recursive definitions decouple unbounded behavior from mobility and are not first-class. The definition $\hat{p} \triangleq a \bullet \hat{p}$ is the nearest we can get to $!(p \setminus a \bullet !p)$, and this definition does not imply mobility. As we will see in chapter 9, recursive definitions will therefore prove to be a good match for recursive processes, which ought not to be able to move around within a configuration just because they have unbounded behavior.

Fortier and Santocanale[24] have used recursive definitions together with circular derivations in a fragment of the linear sequent calculus, establishing a sound condition under which these definitions constitute least and greatest fixed points and these derivations constitute valid inductive and coinductive proofs. Extending their work, Derakhshan and Pfenning[25] have presented a related, locally decidable condition on circular derivations in first-order intuitionistic multiplicative and additive linear logic that ensures cut elimination is productive. We know of no work on applying these ideas to ordered logic, but we expect that similar results ought to hold. In any case, in the remainder

[24] Fortier and Santocanale 2013.

[25] Derakhshan and Pfenning 2020.

of this document we are concerned only with general recursive definitions, not inductive or coinductive definitions.

3.4 *Other extensions*

Ordered logic can also be extended in other directions that we briefly describe here: first-order universal and existential quantifiers, multiplicative falsehood and disjunction, a mobility modality, the aforementioned persistence modality, and subexponentials, and adjunctions from adjoint logic. These extensions are not crucial to the remainder of this dissertation, but are mentioned for the sake of completeness.

Adding first-order universal and existential quantifiers, $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$, to the ordered sequent calculus is completely standard. Sequents are extended with a separate context of well-sorted term variables, $x{:}\tau$; this new context is structural, admitting weakening, contraction, and exchange properties.

Multiplicative falsehood can be introduced into the ordered sequent calculus, as in the intuitionistic linear sequent calculus: by generalizing sequents to allow an empty consequent, $\Omega \vdash (\cdot)$.[26] With this new judgment form, the cut principle and left rules must be revised to allow the empty consequent. Multiplicative falsehood, $\bot$, internalizes this judgment as a proposition and is, as its name suggests, dual to multiplicative truth, $\mathbf{1}$. Multiplicative disjunction, $A \mathbin{⅋} B$, can also be introduced like in the intuitionistic linear sequent calculus; it requires multiple-conclusion sequents.[27]

[26] Chang et al. 2003.

[27] Chang et al. 2003.

In addition to the aforementioned persistence modality, $!A$, it is possible to introduce a mobility modality, $¡A$.[28] Just as persistence is subject to all of the structural properties, $¡A$ is subject to exchange (but neither weakening nor contraction). In this way, $¡A$ represents a mobile resource that may permute with other resources. Related to these modalities are subexponentials[29] and adjunctions from adjoint logic.[30] Both subexponentials and adjunctions allow ordered logic to include multiple distinct layers, each with its own set of structural properties (that must, however, meet certain conditions for cut elimination).

[28] Polakow and Pfenning 1999b.

[29] Nigam and Miller 2009; Kanovich et al. 2019.

[30] Benton 1995; Pruiksma et al. 2018.

Part II

Concurrency as proof construction

4
String rewriting for concurrent specifications

In this chapter, we consider abstract rewriting as a framework for specifying the dynamics of concurrent systems. This is not, of course, a new idea. Multiset rewriting[1] has previously been put forward as a state-transformation model of concurrency, and has been used to describe security protocols,[2] for example. Unlike in multiset rewriting, we are particularly interested in concurrent systems whose components are arranged in a chain topology and have a monoidal structure. Given that finite strings over an alphabet Σ form a free monoid, string rewriting, rather than multiset rewriting, is a good match for the structure we are interested in.

[1] Meseguer 1992; Cervesato and Scedrov 2009.

[2] Cervesato, Durgin, et al. 1999; Durgin et al. 2004.

For a broad sketch of string rewriting, consider the finite strings over the alphabet $\{a, b\}$, and let $\longrightarrow$ be the least compatible binary relation over those strings that satisfies the axioms

$$\frac{}{a\,b \longrightarrow b} \qquad \text{and} \qquad \frac{}{b \longrightarrow \epsilon}\,. \tag{4.1}$$

This relation can be seen as a rewriting relation on strings. For instance, because $a\,b\,b \longrightarrow b\,b$, we would say that $a\,b\,b$ may be rewritten to $b\,b$.

More generally, under the rewriting axioms of eq. (4.1), a string w can be rewritten to the empty string – that is, $w \longrightarrow \cdots \longrightarrow \epsilon$ – if, and only if, that string ends with b. For example, the string abb ends with b, and abb can indeed be rewritten to the empty string:

$$a\,b\,b \longrightarrow b\,b \longrightarrow b \longrightarrow \epsilon\,.$$

In this way, the rewriting axioms of eq. (4.1) constitute a specification of a system that identifies those strings over the alphabet $\{a, b\}$ that end with b.

The usual operational semantics for string rewriting employs committed-choice nondeterminism, which can lead to stuck, or otherwise undesirable, states. For example, although abb certainly ends with b, the string abb can be rewritten to a, a stuck state, if incorrect choices about which axioms to apply are made:

$$a\,b\,b \longrightarrow a\,b \longrightarrow a \not\longrightarrow\,.$$

No backtracking is performed to reconsider these choices.

Disjoint segments of a string may be rewritten independently. For example, the substring $a\,b$ can be rewritten to b, and the final b of $a\,b\,b$ can be rewritten to the empty string. Being independent, these rewritings can occur in either order, as shown in the adjacent figure. Concurrency arises when the various interleavings of independent rewritings are treated indistinguishably.

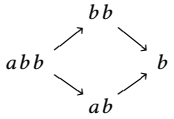


Figure 4.1: The interleavings of two independent rewritings

The remainder of this chapter describes a string rewriting framework in more detail (section 4.1) and examines its properties, most importantly concurrent rewritings. Then we present two extended examples of how string rewriting may be used to specify concurrent systems: nondeterministic finite automata (section 4.2) and binary representations of natural numbers (section 4.3). These will serve as recurring examples throughout the remainder of this document.

4.1 *A string rewriting framework*

In this section, we present a string rewriting framework and examine some of its basic properties.

4.1.1 *Symbols and strings*

String rewriting presupposes an alphabet, Σ, of symbols a from which finite strings are constructed. This alphabet is usually, but need not be, finite.

Strings, w, are then finite lists of symbols: $w = a_1\,a_2 \cdots a_n$. Algebraically, strings form a free (noncommutative) monoid over symbols $a \in \Sigma$ and may be described syntactically by the grammar

$$w ::= w_1\,w_2 \mid \epsilon \mid a\,,$$

where the monoid operation is string concatenation, denoted by $w_1\,w_2$, and the unit element is the empty string, denoted by ϵ.[3]

As a monoid, strings are equivalent up to associativity and unit laws (see adjacent figure). We choose to keep this equivalence implicit, however, treating equivalent strings as syntactically indistinguishable. As usual for a free monoid, the alternative grammar $w ::= \epsilon \mid a\,w$ can be used to describe the same strings.

[3] Strings are isomorphic to the finite words used by automata (chapter 2), but the two serve different conceptual roles in this document.

$$(w_1\,w_2)\,w_3 = w_1\,(w_2\,w_3)$$
$$\epsilon\,w = w = w\,\epsilon$$

Figure 4.2: Monoid laws for strings

4.1.2 *A rewriting relation*

At the heart of string rewriting is a binary relation, $\longrightarrow$, over strings. When $w \longrightarrow w'$, we say that w can be rewritten to w'. This relation is defined as the least compatible relation satisfying a collection of rewriting axioms, chosen on a per-application basis, such as the axioms

$$\frac{}{a\,b \longrightarrow b} \qquad \text{and} \qquad \frac{}{b \longrightarrow \epsilon}$$

shown earlier. More generally, an axiom is any pair of concrete, finite strings, $w \longrightarrow w'$, although axioms of the form $\epsilon \longrightarrow w'$ are expressly forbidden.

To be more formal, these axioms are collected into a signature, Θ, that indexes the rewriting relation:

$$\Theta ::= \cdot \mid \Theta, w \longrightarrow w' \quad (w \neq \epsilon)$$

The axioms of this signature may then be used via a $\longrightarrow$AX rule,

$$\frac{w \longrightarrow w' \in \Theta}{w \longrightarrow_\Theta w'} \longrightarrow \text{AX} .$$

Aside from this rule, all of the other rules for the rewriting relation simply pass on the signature Θ untouched; for this reason, we nearly always elide the signature index on the rewriting relation, writing $\longrightarrow$ instead of $\longrightarrow_\Theta$. As an example signature, the axioms of our running example can be packaged as

$$\begin{aligned} \Sigma &= \{a, b\} \\ \Theta &= (a\,b \longrightarrow b)\,, (b \longrightarrow \epsilon)\,. \end{aligned} \tag{4.2}$$

In addition to the application-specific axioms contained within a signature, rewriting is always permitted within substrings, so we adopt the rule

$$\frac{w_0 \longrightarrow w_0'}{w_1\, w_0\, w_2 \longrightarrow w_1\, w_0'\, w_2} \longrightarrow \text{C}$$

to ensure that the rewriting relation is compatible with the monoidal structure of strings.

The $\longrightarrow$ relation thus describes the rewritings that are possible in a single step: exactly one axiom, perhaps embellished by the compatibility rules. In addition to these single-step rewritings, it will frequently be useful to describe the rewritings that are possible in some finite number of steps. For this, we construct a multi-step rewriting relation, $\Longrightarrow$, from the reflexive, transitive closure of $\longrightarrow$.[4]

[4] Usually written as $\longrightarrow^*$, we instead choose $\Longrightarrow$ for the reflexive, transitive closure because of its similarity with standard process calculus notation for weak transitions, $\overset{\alpha}{\Longrightarrow}$. Our reasons for this choice of notation will become clearer in subsequent chapters.

Consistent with its monoidal structure, there are two equivalent formulations of this reflexive, transitive closure: each rewriting sequence $w \Longrightarrow w'$ can be viewed as either a list or tree of individual rewriting steps. We prefer the list-based formulation,

$$\frac{}{w \Longrightarrow w} \Longrightarrow \text{R} \qquad \text{and} \qquad \frac{w \longrightarrow w' \quad w' \Longrightarrow w''}{w \Longrightarrow w''} \Longrightarrow \text{T} ,$$

because it tends to streamline proofs by structural induction. However, on the basis of the following lemma, we allow ourselves to freely switch between the two formulations as needed.

LEMMA 4.1 (Transitivity of $\Longrightarrow$). *If $w \Longrightarrow w'$ and $w' \Longrightarrow w''$, then $w \Longrightarrow w''$.*

Proof. By structural induction over the first of the given rewriting sequences, $w \Longrightarrow w'$. □

A summary of string rewriting is shown in fig. 4.3.

$$\textsc{strings} \quad w ::= w_1\, w_2 \mid \epsilon \mid a$$

$$\textsc{signatures} \quad \Theta ::= \cdot \mid \Theta, w \longrightarrow w' \quad (w \neq \epsilon)$$

$$(w_1\, w_2)\, w_3 = w_1\, (w_2\, w_3)$$
$$\epsilon\, w = w = w\, \epsilon$$

$$\frac{w \longrightarrow w' \in \Theta}{w \longrightarrow_\Theta w'} \longrightarrow\text{AX} \qquad \frac{w_0 \longrightarrow w_0'}{w_1\, w_0\, w_2 \longrightarrow w_1\, w_0'\, w_2} \longrightarrow\text{C}$$

$$\frac{}{w \Longrightarrow w} \Longrightarrow\text{R} \qquad \frac{w \longrightarrow w' \quad w' \Longrightarrow w''}{w \Longrightarrow w''} \Longrightarrow\text{T}$$

Figure 4.3: A string rewriting framework

4.1.3 *Properties of the string rewriting framework*

As an abstract rewriting system, the above string rewriting framework can be evaluated for several properties: confluence, termination, and, of particular interest to us, concurrency.

CONCURRENCY As an example multi-step rewriting sequence, observe that $a\,b\,b \Longrightarrow \epsilon$, under the axioms of our running example (eq. (4.2)). In fact, as shown in the adjacent figure, multiple sequences witness this rewriting. The initial $a\,b$ can first be rewritten to b and then the terminal b can be rewritten to ϵ (upper half of figure); or vice versa: the terminal b can first be rewritten to ϵ and then the initial $a\,b$ can be rewritten to b (lower half of figure). In either case, the remaining b (which is the leftmost of the original bs) can finally be rewritten to ϵ.

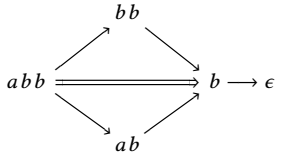


Figure 4.4: An example of concurrent string rewriting

Notice that these two sequences differ only in how non-overlapping, and therefore independent, rewritings of the string's two segments are interleaved. Consequently, the two sequences can be – and indeed should be – considered essentially equivalent. The details of how the individual, small steps are interleaved are irrelevant, so that – conceptually at least – only the big-step sequence from $a\,b\,b$ to b (and ultimately ϵ) remains (middle of figure).

In contrast, a third rewriting sequence does not admit this reordering: the leftmost b is rewritten first to ϵ and then the resulting $a\,b$ is rewritten to b (and ultimately ϵ). This sequence's two rewriting steps are not independent because the b that participates in the rewriting of $a\,b$ is not adjacent to the a until after the first rewriting step occurs. This is captured in the adjacent figure by distinguishing the two bs with subscripts.

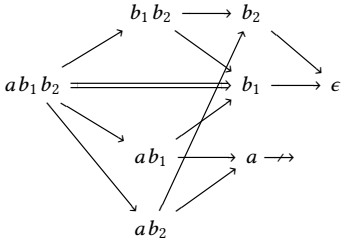


Figure 4.5: When multiple occurrences of b are properly distinguished, a complete trace diagram can be given.

More generally, this idea that the interleaving of independent actions is irrelevant is known as *concurrent equality*,[5] and it forms the basis of concurrency. With the partial commutativity endowed by concurrent equality,

[5] Watkins et al. 2002.

the free monoid formed by rewriting sequences is, more specifically, a trace monoid. As such, we will frequently refer to rewriting sequences as *traces*.

NON-CONFLUENCE We may also evaluate string rewriting for confluence. Confluence requires that all strings with a common ancestor be joinable, *i.e.*, that $w_1' \Longleftrightarrow w_2'$ implies $w_1' \Longrightarrow \Longleftarrow w_2'$, for all strings w_1' and w_2'.

String rewriting is an asymmetric, committed-choice relation, so some nondeterministic choices are irreversible. For example, under the axioms of our running example (eq. (4.2)), $a\,b$ can be nondeterministically rewritten into either a or ϵ, as shown in fig. 4.5. However, neither a nor ϵ can be rewritten, so confluence fails to hold for string rewriting in general.

NON-TERMINATION In our running example, rewriting always terminates: each possible rewriting step removes exactly one symbol, and each string contains only finitely many symbols.

In general, string rewriting does not terminate even though strings are finite. For a simple example, consider rewriting of strings over the alphabet $\{a, b\}$ with axioms $a \longrightarrow b$ and $b \longrightarrow a$. Every finite trace from a nonempty string can always be extended by applying one of these axioms, so string rewriting in this example never terminates.

4.2 *Example: Nondeterministic finite automata*

As an extended example of string rewriting, we will specify how an NFA processes its input. Beginning with this specification, NFAs will serve as a recurring example throughout the remainder of this document.

Given an NFA $\mathcal{A} = (Q, \Delta, F)$ over an input alphabet Σ, the idea is to introduce a string rewriting axiom for each transition that the NFA can make:

$$\overline{a\,q \longrightarrow q_a'} \text{ for each transition } q \xrightarrow{a} q_a'.$$

In addition, the NFA's acceptance criteria is captured by introducing a distinguished symbol \$ to act as an end-of-word marker, along with axioms

$$\overline{\$\,q \longrightarrow F(q)} \text{ for each state } q, \text{ where } F(q) = \begin{cases} y & \text{if } q \in F \\ n & \text{if } q \notin F\,. \end{cases}$$

These axioms imply that rewriting occurs over the finite strings from $\{\$\} \times \Sigma^* \times Q \cup \{y, n\}$. Expressed as a string rewriting signature, the NFA $\mathcal{A}$ is

$$\Theta = \{a\,q \longrightarrow q_a' \mid q \xrightarrow{a} q_a'\} \cup \{\$\,q \longrightarrow F(q) \mid q \in Q\}\,,$$

where $F(q)$ is defined as above.

For a concrete instance of this encoding, recall from chapter 2 the NFA (repeated in the adjacent figure) that accepts exactly those words, over the

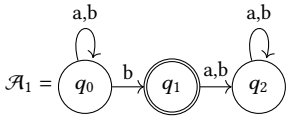


Figure 4.6: An NFA that accepts, from state q_0, exactly those words that end with b. (Repeated from fig. 2.1.)

alphabet $\Sigma = \{a, b\}$, that end with b; that NFA is specified by the following string rewriting axioms:

$$\begin{aligned}\Theta_{\mathcal{A}_1} = (a\,q_0 \longrightarrow q_0)\,, (b\,q_0 \longrightarrow q_0)\,, (b\,q_0 \longrightarrow q_1)\,, (\$\,q_0 \longrightarrow n)\,,\\ (a\,q_1 \longrightarrow q_2)\,, \quad (b\,q_1 \longrightarrow q_2) \quad\quad , (\$\,q_1 \longrightarrow y)\,,\\ (a\,q_2 \longrightarrow q_2)\,, \quad (b\,q_2 \longrightarrow q_2) \quad\quad , (\$\,q_2 \longrightarrow n)\,.\end{aligned}$$

Indeed, just as the NFA $\mathcal{A}_1$ accepts the input word abb, its rewriting specification admits a trace

$$\$\,b\,b\,a\,q_0 \longrightarrow \$\,b\,b\,q_0 \longrightarrow \$\,b\,q_0 \longrightarrow \$\,q_1 \longrightarrow y\,.$$

More generally, this string rewriting specification of NFAs adequately describes their operational semantics, in the sense that it simulates all NFA transitions. Given the reversal (anti-)homomorphism for finite words defined in the adjacent figure, we can prove the following adequacy result.

$$\begin{aligned}(w_1\,w_2)^{\mathrm{R}} &= w_2^{\mathrm{R}}\,w_1^{\mathrm{R}}\\ \epsilon^{\mathrm{R}} &= \epsilon\\ a^{\mathrm{R}} &= a\end{aligned}$$

Figure 4.7: An (anti-)homomorphism for reversal of finite words

Theorem 4.2 (Adequacy of NFA specification). *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ.*

- *$q \xrightarrow{a} q'_a$ if, and only if, $a\,q \longrightarrow q'_a$, for all input symbols $a \in \Sigma$.*
- *$q \in F$ if, and only if, $\$\,q \longrightarrow y$.*
- *$q \xrightarrow{w} q'$ if, and only if, $w^{\mathrm{R}}\,q \Longrightarrow q'$, for all finite words $w \in \Sigma^*$.*

Proof. The first two parts follow immediately from the NFA's string rewriting specification; the third part follows by induction over the structure of the input word w. □

This adequacy theorem is relatively straightforward to state and prove because string rewriting is a good match for labeled transition systems, like the one that defines an NFA's operational semantics. On the other hand, when a system is not so clearly based on a labeled transition system, stating and proving the adequacy of its string rewriting specification becomes a bit more involved. This is the case for the next example, binary representations of natural numbers.

4.3 *Example: Binary representations of natural numbers*

For a second recurring example, we will use binary representations of natural numbers equipped with increment and decrement operations. Here we present a string rewriting specification of these *binary counters*.

4.3.1 *Binary representations*

In this setting, we represent a natural number in binary by a string that consists of a big-endian sequence of symbols b_0 and b_1, prefixed by the symbol e; leading b_0s are permitted. For example, both $w = e\,b_1$ and $w' = e\,b_0\,b_1$ are valid binary representations of the natural number 1.

To be more precise, we inductively define a relation, $\approx_{\mathrm{v}}$, that assigns to each binary representation a unique natural number denotation. If $w \approx_{\mathrm{v}} n$, we say that w denotes, or represents, natural number n in binary.

$$\frac{}{e \approx_{\mathrm{v}} 0}\ e\text{-v} \qquad \frac{w \approx_{\mathrm{v}} n}{w\,b_0 \approx_{\mathrm{v}} 2n}\ b_0\text{-v} \qquad \frac{w \approx_{\mathrm{v}} n}{w\,b_1 \approx_{\mathrm{v}} 2n+1}\ b_1\text{-v}$$

Besides providing a denotational semantics of binary numbers, the $\approx_{\mathrm{v}}$ relation also serves to implicitly characterize the well-formed binary numbers as those strings w that form the relation's domain of definition.[6]

The adequacy of the $\approx_{\mathrm{v}}$ relation is proved as the following theorem.

Theorem 4.3 (Adequacy of binary representations). *Binary representations and their $\approx_{\mathrm{v}}$ relation are:*

Functional *For each binary number w, there exists a unique natural number n such that $w \approx_{\mathrm{v}} n$.*

Surjective *For each natural number n, there exists a binary number w such that $w \approx_{\mathrm{v}} n$.*

Latent *If $w \approx_{\mathrm{v}} n$, then $w \not\longrightarrow$.*

Proof. The three claims may be proved by induction over the structure of w, and by induction on n, respectively. □

Notice that the above e-v and b_0-v rules overlap when the denotation is 0, giving rise to the leading b_0s that make the $\approx_{\mathrm{v}}$ relation non-injective: for example, both $e\,b_1 \approx_{\mathrm{v}} 1$ and $e\,b_0\,b_1 \approx_{\mathrm{v}} 1$ hold. However, if the b_0-v is restricted to *nonzero* even numbers, then each natural number has a unique, canonical representation that is free of leading b_0s.[7]

4.3.2 *An increment operation*

To use string rewriting to describe an increment operation on binary representations, we introduce a new symbol, i, that will serve as an increment instruction.

Given a binary number w that represents n, we may append i to form an active[8], computational string, $w\,i$. For i to adequately represent the increment operation, the string $w\,i$ must meet two conditions, captured by the following global desiderata:

- $w\,i \Longrightarrow\!\approx_{\mathrm{v}} n+1$ – that is, *some* rewriting sequence results in a binary representation of $n+1$; and
- $w\,i \Longrightarrow w'$ implies $w' \Longrightarrow\!\approx_{\mathrm{v}} n+1$ – that is, *any* rewriting sequence from $w\,i$ can result in a binary representation of $n+1$.

For example, because $e\,b_1$ denotes 1, a computation $e\,b_1\,i \Longrightarrow\!\approx_{\mathrm{v}} 2$ must exist; moreover, every computation $e\,b_1\,i \Longrightarrow\!\approx_{\mathrm{v}} n'$ must satisfy $n' = 2$.

[6] Alternatively, the well-formed binary numbers could be described more explicitly by the grammar

$$w ::= e \mid w\,b_0 \mid w\,b_1\,,$$

and then their denotations could be expressed in a more functional manner:

$$[\![e]\!]_{\mathrm{v}} = 0$$
$$[\![w\,b_0]\!]_{\mathrm{v}} = 2[\![w]\!]_{\mathrm{v}}$$
$$[\![w\,b_1]\!]_{\mathrm{v}} = 2[\![w]\!]_{\mathrm{v}} + 1\,.$$

We prefer the purely relational formulation, however.

[7] A restriction of the b_0 rule to nonzero even numbers is:

$$\frac{w \approx_{\mathrm{v}} n \quad (n > 0)}{w\,b_0 \approx_{\mathrm{v}} 2n}\,.$$

The leading-b_0-free representations could alternatively be seen as the canonical representatives of the equivalence classes induced by the relation among binary numbers that have the same denotation: $w \equiv w'$ if $w \approx_{\mathrm{v}} n$ and $w' \approx_{\mathrm{v}} n$ for some n.

[8] The 'active', 'latent', and 'passive' terminology is borrowed from Pfenning and Simmons (2009). Active strings are immediately rewritable, but latent strings are rewritable only when combined with other, passive strings. The blurry line between latent and passive strings is exploited in chapter 6 when we discuss choreographies.

To achieve these global desiderata, we introduce three string rewriting axioms that describe how the symbols e, b_0, and b_1 may be rewritten when they encounter i, the increment instruction:

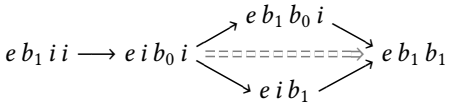
$$\overline{e\ i \longrightarrow e\ b_1} \qquad \overline{b_0\ i \longrightarrow b_1} \qquad \text{and} \qquad \overline{b_1\ i \longrightarrow i\ b_0}\,.$$

These three axioms can be read as follows:

- To increment e, replace e (and i) with $e\ b_1$.
- To increment a binary number ending in b_0, flip that bit to b_1.
- To increment a binary number ending in b_1, flip that bit to b_0 and carry the increment over to the more significant bits.

Comfortingly, $1 + 1 = 2$: a trace $e\ b_1\ i \longrightarrow e\ i\ b_0 \longrightarrow e\ b_1\ b_0$ indeed exists.

Owing to the notion of concurrent equality that string rewriting admits, increments may even be performed concurrently. For example, there are two rewriting sequences that witness $e\ b_1\ i\ i \Longrightarrow e\ b_1\ b_1$:



In other words, once the left most increment is carried past the least significant bit, the two increments can be interleaved, with no observable difference in the outcome.

These increment axioms introduce strings that occur as intermediate computational states within traces, such as $e\ i\ b_0\ i$ and $e\ i\ b_1$ in the above diagram. To characterize the valid intermediate strings, we define a binary relation, $\approx_{\text{I}}$, that assigns a natural number denotation to each such intermediate string, not only to the terminal values, as $\approx_{\text{V}}$ did.[9]

$$\frac{}{e \approx_{\text{I}} 0}\ e\text{-I} \qquad \frac{w \approx_{\text{I}} n}{w\ b_0 \approx_{\text{I}} 2n}\ b_0\text{-I} \qquad \frac{w \approx_{\text{I}} n}{w\ b_1 \approx_{\text{I}} 2n+1}\ b_1\text{-I} \qquad \frac{w \approx_{\text{I}} n}{w\ i \approx_{\text{I}} n+1}\ i\text{-I}$$

Binary values should themselves be valid, terminal computational states, so the first three rules are carried over from the $\approx_{\text{V}}$ relation. The i-I rule allows multiple increment instructions to be interspersed throughout the state.

With this $\approx_{\text{I}}$ relation in hand, we can now prove a stronger, small-step adequacy theorem. This small-step theorem then implies the big-step desiderata from above.

Theorem 4.4 (Small-step adequacy of increments).
Value soundness *If $w \approx_{\text{V}} n$, then $w \approx_{\text{I}} n$ and $w \not\longrightarrow$.*
Preservation *If $w \approx_{\text{I}} n$ and $w \longrightarrow w'$, then $w' \approx_{\text{I}} n$.*
Progress *If $w \approx_{\text{I}} n$, then either: $w \longrightarrow w'$ for some w'; or $w \approx_{\text{V}} n$.*
Termination *If $w \approx_{\text{I}} n$, then every rewriting sequence from w is finite.*

[9] Like the $\approx_{\text{V}}$ relation does for values, the $\approx_{\text{I}}$ relation also serves to implicitly characterize the valid intermediate states as those contexts that form the relation's domain of definition. As with values, the valid intermediate states could also be enumerated more explicitly and syntactically with a grammar and denotation function:

$$w ::= e \mid w\,b_0 \mid w\,b_1 \mid w\,i$$

$$[\![e]\!]_{\text{I}} = 0$$
$$[\![w\,b_0]\!]_{\text{I}} = 2[\![w]\!]_{\text{I}}$$
$$[\![w\,b_1]\!]_{\text{I}} = 2[\![w]\!]_{\text{I}} + 1$$
$$[\![w\,i]\!]_{\text{I}} = [\![w]\!]_{\text{I}} + 1$$

However, we once again prefer the purely relational form.

Proof. Each part is proved separately.

Value soundness can be proved by structural induction on the derivation of $w \approx_{\mathrm{V}} n$.

Preservation and progress can likewise be proved by structural induction on the derivation of $w \approx_{\mathrm{I}} n$.

Termination can be proved using an explicit termination measure, $|-|_{\mathrm{I}}$, that is strictly decreasing across each rewriting, $w \longrightarrow w'$. Specifically, we use a measure (see the adjacent figure), adapted from the standard amortized constant work analysis of increment for binary counters.[10] The measure $|-|_{\mathrm{I}}$ is such that $w \longrightarrow w'$ implies $|w|_{\mathrm{I}} > |w'|_{\mathrm{I}}$; because the measure is always nonnegative, only finitely many such rewritings can occur.

As an example case, consider the intermediate state $w\, b_1\, i$ and its rewriting $w\, b_1\, i \longrightarrow w\, i\, b_0$. Indeed, $|w\, b_1\, i|_{\mathrm{I}} = |w|_{\mathrm{I}} + 3 > |w|_{\mathrm{I}} + 2 = |w\, i\, b_0|_{\mathrm{I}}$. □

$$
\begin{aligned}
|e|_{\mathrm{I}} &= 0 \\
|w\, b_0|_{\mathrm{I}} &= |w|_{\mathrm{I}} \\
|w\, b_1|_{\mathrm{I}} &= |w|_{\mathrm{I}} + 1 \\
|w\, i|_{\mathrm{I}} &= |w|_{\mathrm{I}} + 2
\end{aligned}
$$

Figure 4.8: A termination measure, adapted from the standard amortized work analysis of increment for binary counters

[10] Cormen et al. 2009.

COROLLARY 4.5 (Big-step adequacy of increments).

Evaluation *If $w \approx_{\mathrm{I}} n$, then $w \Longrightarrow \approx_{\mathrm{V}} n$. In particular, if $w \approx_{\mathrm{V}} n$, then $w\, i \Longrightarrow \approx_{\mathrm{V}} n + 1$.*

Preservation *If $w \approx_{\mathrm{I}} n$ and $w \Longrightarrow w'$, then $w' \approx_{\mathrm{I}} n$. In particular, if $w \approx_{\mathrm{V}} n$ and $w\, i \Longrightarrow w'$, then $w' \Longrightarrow \approx_{\mathrm{V}} n + 1$.*

Proof. The two parts are proved separately.

Evaluation can be proved by repeatedly appealing to the progress and preservation results (theorem 4.4). By the accompanying termination result, a binary value must eventually be reached.

Preservation can be proved by structural induction on the given trace. □

4.3.3 *A decrement operation*

Binary counters may also be equipped with a decrement operation. Instead of examining decrements *per se*, we will describe a very closely related operation: the normalization of binary representations to what might be called *head-unary form*. (We will frequently abuse terminology, using 'head-unary normalization' and 'decrement operation' interchangeably.) A string w will be said to be in head-unary form if it has one of two forms: $w = z$; or $w = w'\, s$, for some binary number w'.

Just as appending the symbol i to a counter w initiates an increment, appending a symbol d will cause the counter to begin normalizing to head-unary form. For d to adequately represent this operation, the string $w\, d$ must satisfy the following global desiderata when $w \approx_{\mathrm{D}} n$:

- $w\, d \Longrightarrow z$ if, and only if, $n = 0$;
- $w\, d \Longrightarrow w'\, s$ for some w' such that $w' \approx_{\mathrm{V}} n - 1$, if $n > 0$; and
- $w\, d \Longrightarrow w'\, s$ only if $n > 0$ and $w' \approx_{\mathrm{V}} n - 1$.

For example, because $e\,b_1$ denotes 1, a trace $e\,b_1\,d \implies w'\,s$ must exist, for some $w' \approx_{\mathrm{V}} 0$.

To achieve these global desiderata, we introduce three additional axioms that describe how the symbols e, b_0, and b_1 may be rewritten when they encounter d, the decrement instruction; also, an intermediate symbol b_0' and two more axioms are introduced:

$$\overline{e\,d \longrightarrow z} \qquad \overline{b_1\,d \longrightarrow b_0\,s} \qquad \overline{b_0\,d \longrightarrow d\,b_0'}$$

$$\overline{z\,b_0' \longrightarrow z} \qquad \text{and} \qquad \overline{s\,b_0' \longrightarrow b_1\,s}\,.$$

These five axioms can be read as follows:

- Because e denotes 0, its head-unary form is simply z.
- Because $w\,b_1$ denotes $2n+1$ if w denotes n, its head-unary form, $w\,b_0\,s$, can be constructed by flipping the least significant bit to b_0 and appending s.
- Because $w\,b_0$ denotes $2n$ if w denotes n, its head-unary form can be constructed by recursively putting the more significant bits, w, into head-unary form and appending b_0' to process that result.
  - If w has head-unary form z and therefore denotes 0, then $w\,b_0$ also denotes 0 and has head-unary form z.
  - Otherwise, if w has head-unary form $w'\,s$ and thus denotes $n > 0$, then $w\,b_0$ denotes $2n > 0$ and has head-unary form $w'\,b_1\,s$, which can be constructed by replacing s with $b_1\,s$.

Comfortingly, $(1 + 1) - 1 = 1$: the head-unary form of $e\,b_1\,i$ is $e\,b_0\,b_1\,s$:

$$e\,b_1\,i\,d \longrightarrow e\,i\,b_0\,d \begin{array}{c} \nearrow \\ \Longrightarrow \\ \searrow \end{array} \begin{array}{l} e\,b_1\,b_0\,d \\ \\ e\,i\,d\,b_0' \end{array} \begin{array}{c} \searrow \\ \\ \nearrow \end{array} e\,b_1\,d\,b_0' \longrightarrow e\,b_0\,s\,b_0' \longrightarrow e\,b_0\,b_1\,s\,.$$

Note the concurrency that derives from the independence of the increment and decrement after the initial step of rewriting.

These decrement axioms introduce more strings that may occur as intermediate computational states. As before, we define a new binary relation, $\approx_{\mathrm{D}}$, that assigns a natural number denotation to each string that may appear as an intermediate state during a decrement.

$$\frac{w \approx_{\mathrm{I}} n}{w\,d \approx_{\mathrm{D}} n}\ d\text{-D} \qquad \frac{w \approx_{\mathrm{D}} n}{w\,b_0' \approx_{\mathrm{D}} 2n}\ b_0'\text{-D} \qquad \frac{}{z \approx_{\mathrm{D}} 0}\ z\text{-D} \qquad \frac{w \approx_{\mathrm{I}} n}{w\,s \approx_{\mathrm{D}} n+1}\ s\text{-D}$$

At first glance, the d-D rule may look a bit odd: Why is the denotation unchanged by a decrement, $w\,d$? Because the operation is more accurately characterized as head-unary normalization, it makes sense that the denotation remains unchanged. The operation described by d does not change the binary counter's value – it only expresses that same value in a different form.[11]

[11] Once again, the valid intermediate states could also be enumerated more explicitly and syntactically with a grammar and denotation function:

$$w ::= e \mid w\,b_0 \mid w\,b_1 \mid w\,i$$
$$w^h ::= w\,d \mid w^h\,b_0' \mid z \mid w\,s$$
$$[\![w\,d]\!]_{\mathrm{D}} = [\![w]\!]_{\mathrm{I}}$$
$$[\![w^h\,b_0']\!]_{\mathrm{D}} = 2[\![w^h]\!]_{\mathrm{D}}$$
$$[\![z]\!]_{\mathrm{D}} = 0$$
$$[\![w\,s]\!]_{\mathrm{D}} = [\![w]\!]_{\mathrm{I}} + 1$$

Also, notice that the premises of the d-D and s-D rules use the increment-only denotation relation, $\approx_{\mathrm{I}}$, not the decrement relation, $\approx_{\mathrm{D}}$. These choices ensure that each counter has at most one d and may not have any i or s symbols to the right of that d. But the premise of the b_0'-D does use the $\approx_{\mathrm{D}}$ relation, so d may have b_0' symbols to its right.

With this $\approx_{\mathrm{D}}$ relation in hand, we can now prove a small-step adequacy theorem. This small-step theorem then implies the big-step desiderata from above.

THEOREM 4.6 (Small-step adequacy of decrements).
Preservation *If $w \approx_{\mathrm{D}} n$ and $w \longrightarrow w'$, then $w' \approx_{\mathrm{D}} n$.*
Progress *If $w \approx_{\mathrm{D}} n$, then either:*
- *$w \longrightarrow w'$, for some w';*
- *$n = 0$ and $w = z$; or*
- *$n > 0$ and $w = w'\,s$, for some w' such that $w' \approx_{\mathrm{I}} n - 1$.*

Termination *If $w \approx_{\mathrm{D}} n$, then every rewriting sequence from w is finite.*

Proof. Each part is proved separately.

Preservation and progress are proved, as before, by structural induction on the given derivation of $w \approx_{\mathrm{D}} n$.

Termination is proved by exhibiting a measure, $|-|_{\mathrm{D}}$, given in the adjacent figure, that is strictly decreasing across each rewriting. Unlike the amortized constant work increments (see proof of theorem 4.4), this measure assigns a linear amount of potential to the decrement instruction.[12]

This measure is strictly decreasing across each rewriting: $w \longrightarrow w'$ only if $|w|_{\mathrm{D}} > |w'|_{\mathrm{D}}$. As an example case, consider the intermediate state $w\,b_0\,d$ and its rewriting $w\,b_0\,d \longrightarrow w\,d\,b_0'$. Indeed,

$$|w\,b_0\,d|_{\mathrm{D}} = |w|_{\mathrm{I}} + 3|w| + 3 > |w|_{\mathrm{I}} + 3|w| + 2 = |w\,d\,b_0'|_{\mathrm{D}}\,. \qquad \square$$

$$|w\,d|_{\mathrm{D}} = |w|_{\mathrm{I}} + 3|w|$$
$$|w\,b_0'|_{\mathrm{D}} = |w|_{\mathrm{D}} + 2$$
$$|z|_{\mathrm{D}} = 0$$
$$|w\,s|_{\mathrm{D}} = |w|_{\mathrm{I}}$$

Figure 4.9: A termination measure for decrements, where $|w|$ denotes the length of string w

[12] Actually, because the increment and decrement operations are defined only for binary representations, not head-unary forms, there can be at most one d. Therefore, it is actually possible to assign a constant amount of potential to each d. However, doing so would rely on a somewhat involved lexicographic measure that isn't particularly relevant to our aims in this dissertation, so we use the simpler linear potential.

COROLLARY 4.7 (Big-step adequacy of decrements). *If $w \approx_{\mathrm{D}} n$, then:*
- *$w \Longrightarrow z$ if, and only if, $n = 0$;*
- *$w \Longrightarrow w'\,s$ for some w' such that $w' \approx_{\mathrm{I}} n - 1$, if $n > 0$; and*
- *$w \Longrightarrow w'\,s$ only if $n > 0$ and $w' \approx_{\mathrm{I}} n - 1$.*

Proof. From the small-step preservation result of theorem 4.6, it is possible to prove, using a structural induction on the given trace, a big-step preservation result: namely, that $w \approx_{\mathrm{D}} n$ and $w \Longrightarrow w'$ only if $w' \approx_{\mathrm{D}} n$. Each of the above claims then follows from either progress and termination (theorem 4.6) or big-step preservation together with inversion. $\square$

5
Ordered rewriting

The previous chapter reviewed a string rewriting framework for specifying the dynamics of concurrent systems that have chain topologies. As seen in section 4.1.3, string rewriting is indeed a model of concurrency. But it does not have obvious connections to proof construction.

In this chapter, we therefore turn our attention to developing a rewriting interpretation of the ordered sequent calculus shown in chapter 3. Ordered rewriting increases the expressive power of string rewriting by enriching the rather spartan free monoidal structure to one based on free residuated lattices, in accordance with the (full) Lambek calculus.[1] Like the string rewriting framework that it generalizes, ordered rewriting will serve as a framework for specifying concurrent systems that have chain topologies. But unlike string rewriting, ordered rewriting, being derived from the ordered sequent calculus, will have the advantage of constituting a *proof-construction* explanation of concurrency.

[1] Lambek 1958, 1961; Abrusci 1990; Kanazawa 1992; Polakow and Pfenning 1999b.

Section 5.1 begins by observing that many of the ordered sequent calculus's left rules share a large amount of boilerplate, with only very little of each left rule being devoted to the primary task of decomposing the principal proposition. By introducing a new judgment, $\Omega \longrightarrow \Omega'$, for decomposing principal propositions, it is possible to refactor the ordered sequent calculus in such a way that boilerplate is almost exclusively confined to a cut principle for the decomposition judgment (section 5.1.1). Ordered rewriting is then obtained as exactly the decomposition-centric fragment of the refactored sequent calculus (section 5.1.2). To the best of our knowledge, this kind of refactoring appears to be a new way of deriving a rewriting framework from existing proof theory.

Like its string rewriting cousin, the ordered rewriting framework allows disjoint segments of an ordered context to be rewritten independently; concurrency arises within ordered rewriting when the various interleavings of independent rewritings are treated indistinguishably (section 5.1.3).

Unfortunately, this ordered rewriting framework is based on very fine-grained decomposition, which can lead to rewriting sequences that may get

stuck in undesirable and unintended ways. So section 5.2 extends ordered rewriting with ideas from focusing,[2] specifically the higher-order formulation of focusing,[3] to allow for coarser-grained steps of decomposition. These coarser decompositions will be better suited to specifying the dynamics of concurrent systems – rewriting can still get stuck, but no longer in artificial ways.

Moreover, in moving to focused ordered rewriting, no expressive power is lost. With careful placement of shifts, it is possible to control the behavior of focused rewriting, as section 5.3 shows. In particular, unfocused ordered rewriting can be recovered in an operationally faithful way within this focused framework (section 5.3.1), and even an intermediate, weakly focused form of ordered rewriting can be embedded (section 5.3.2).

[2] Andreoli 1992.

[3] Zeilberger 2008; Simmons 2012.

5.1 *Ordered resource decomposition as rewriting*

5.1.1 *Most left rules decompose ordered resources*

Recall two of the ordered sequent calculus's left rules, the $\bullet\text{L}$ and $\&\text{L}_1$ rules shown in the margin. Both rules decompose the principal resource: in the $\bullet\text{L}$ rule, $A \bullet B$ into the separate resources $A\,B$; and, in the $\&\text{L}_1$ rule, $A \mathbin{\&} B$ into A. However, in both cases, the resource decomposition is somewhat obscured by boilerplate. The framed contexts Ω'_L and Ω'_R and goal C serve to enable the rules to be applied anywhere in the list of resources, without restriction; these concerns are not specific to the $\bullet\text{L}$ and $\&\text{L}_1$ rules, but are general boilerplate that arguably should be factored out. Let us develop a variant of the sequent calculus that factors out this boilerplate.

$$\frac{\Omega'_L\ A\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash C}\ \bullet\text{L}$$

and

$$\frac{\Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ (A \mathbin{\&} B)\ \Omega'_R \vdash C}\ \&\text{L}_1$$

To decouple the resource decomposition from the surrounding boilerplate, we will introduce a new judgment, $\Omega \longrightarrow \Omega'$, meaning "Resources Ω may be decomposed into resources Ω'." The choice of notation for this judgment is not coincidental: resource decomposition is, in some dimensions, a generalization of the string rewriting shown in chapter 4.

With this new decomposition judgment comes a cut principle, $\text{CUT}^{\longrightarrow}$, into which all of the boilerplate is factored:

$$\frac{\Omega \longrightarrow \Omega' \quad \Omega'_L\ \Omega'\ \Omega'_R \vdash C}{\Omega'_L\ \Omega\ \Omega'_R \vdash C}\ \text{CUT}^{\longrightarrow}.$$

Using this cut principle, the standard left rules can then be recovered from resource decomposition rules. For example, the decomposition of $A \bullet B$ into $A\,B$ is captured by

$$\frac{}{A \bullet B \longrightarrow A\,B}\ \bullet\text{D},$$

and the standard $\bullet\text{L}$ rule can then be recovered as shown in the adjacent figure. The left rules for $\mathbf{1}$ and $A \mathbin{\&} B$ can be similarly refactored into the resource decomposition rules

$$\frac{}{\mathbf{1} \longrightarrow \cdot}\ \mathbf{1}\text{D} \qquad \frac{}{A \mathbin{\&} B \longrightarrow A}\ \&\text{D}_1 \qquad \text{and} \qquad \frac{}{A \mathbin{\&} B \longrightarrow B}\ \&\text{D}_2.$$

$$\frac{\Omega'_L\ A\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash C}\ \bullet\text{L}$$

$$\rightsquigarrow$$

$$\frac{\dfrac{}{A \bullet B \longrightarrow A\,B}\ \bullet\text{D} \qquad \Omega'_L\ A\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash C}\ \text{CUT}^{\longrightarrow}$$

Figure 5.1: Refactoring the $\bullet\text{L}$ rule in terms of resource decomposition

Even the left rules for left- and right-handed implications can be refactored in this way, despite the additional, minor premises that those rules carry. To keep the correspondence between resource decomposition rules and left rules as close as possible, we could introduce the decomposition rules

$$\frac{\Omega \vdash A}{\Omega\,(A \setminus B) \longrightarrow B}\ \setminus\textsc{d}' \qquad \text{and} \qquad \frac{\Omega \vdash A}{(B \,/\, A)\,\Omega \longrightarrow B}\ /\textsc{d}'. \tag{5.1}$$

Just as for ordered conjunction, the left rules for left- and right-handed implication would then be recoverable via the $\textsc{cut}^{\longrightarrow}$ rule (see adjacent figure).

$$\frac{\Omega \vdash A \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \setminus\textsc{l} \qquad \leftrightsquigarrow \qquad \frac{\dfrac{\Omega \vdash A}{\Omega\,(A \setminus B) \longrightarrow B}\ \setminus\textsc{d}' \qquad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \textsc{cut}^{\longrightarrow}$$

Figure 5.2: A possible refactoring of the $\setminus\textsc{l}$ rule in terms of resource decomposition

Although these rules keep the correspondence between resource decomposition rules and left rules close, they differ from the other decomposition rules in two significant ways. First, the above $\setminus\textsc{d}'$ and $/\textsc{d}'$ rules have premises, and those premises create a dependence of the decomposition judgment upon general provability. Second, the above $\setminus\textsc{d}'$ and $/\textsc{d}'$ rules do not decompose the principal proposition into *immediate* subformulas since Ω is involved. This contrasts with, for example, the $\bullet\textsc{d}$ rule that decomposes $A \bullet B$ into the immediate subformulas $A\,B$.

For these reasons, the above $\setminus\textsc{d}'$ and $/\textsc{d}'$ rules are somewhat undesirable. Fortunately, there is an alternative. Filling in the $\Omega \vdash A$ premises with the $\textsc{id}^A$ rule, we arrive at the derivable rules

$$\frac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\textsc{d} \qquad \text{and} \qquad \frac{}{(B \,/\, A)\,A \longrightarrow B}\ /\textsc{d}, \tag{5.2}$$

which we adopt as decomposition rules in place of those in eq. (5.1). The standard $\setminus\textsc{l}$ and $/\textsc{l}$ rules can still be recovered from these more specific decomposition rules, thanks to $\textsc{cut}$ (see adjacent figure). These revised, nullary decomposition rules correct the earlier drawbacks: like the other decomposition rules, they now have no premises and only refer to immediate subformulas. Moreover, these rules have the advantage of matching two of the axioms from Lambek's original article.[4]

$$\frac{\Omega \vdash A \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \setminus\textsc{l} \qquad \leftrightsquigarrow \qquad \frac{\Omega \vdash A \qquad \dfrac{\dfrac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\textsc{d} \qquad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\, A\, (A \setminus B)\, \Omega'_R \vdash C}\ \textsc{cut}^{\longrightarrow}}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \textsc{cut}^{A}$$

Figure 5.3: Refactoring the $\setminus\textsc{l}$ rule in terms of resource decomposition, via $\setminus\textsc{d}$ and $\textsc{cut}^{\longrightarrow}$

[4] Lambek 1958.

For most ordered logical connectives, this approach works perfectly. Unfortunately, the left rules for additive disjunction, $A \oplus B$, and its unit, $\mathbf{0}$, are resistant to this kind of refactoring. The difficulty with additive disjunction isn't that its left rule, $\oplus\textsc{l}$, doesn't decompose the resource $A \oplus B$. The $\oplus\textsc{l}$ rule certainly does decompose $A \oplus B$, but it does so by branching on the two possible futures: unwrapping the $A \oplus B$ package will result in either A or B. Just as $A \oplus B$ has two futures, $\mathbf{0}$, as its nullary analogue, has no future.

$$\frac{\Omega'_L\, A\, \Omega'_R \vdash C \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\,(A \oplus B)\,\Omega'_R \vdash C}\ \oplus\textsc{l}$$

The straight-line nature of the $\Omega \longrightarrow \Omega'$ judgment seems incompatible with this branching behavior, so we choose to retain the standard $\oplus\textsc{l}$ and $\mathbf{0}\textsc{l}$ rules.[5]

[5] It might be possible to introduce a branching judgment, something like $\Omega \longrightarrow \Omega'_1 \mid \Omega'_2$, but we will not pursue that here.

Figure 5.4 presents the refactored sequent calculus for ordered logic in its entirety. This calculus is sound and complete with respect to the ordered sequent calculus (fig. 3.2).

$$\frac{\Omega \vdash A \quad \Omega'_L\, A\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^A \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{\Omega \longrightarrow \Omega' \quad \Omega'_L\, \Omega'\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^{\longrightarrow}$$

$$\frac{\Omega_1 \vdash A \quad \Omega_2 \vdash B}{\Omega_1\, \Omega_2 \vdash A \bullet B}\ \bullet\textsc{r} \qquad \frac{}{A \bullet B \longrightarrow A\, B}\ \bullet\textsc{d}$$

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}\textsc{r} \qquad \frac{}{\mathbf{1} \longrightarrow \cdot}\ \mathbf{1}\textsc{d}$$

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \,\&\, B}\ \&\textsc{r} \qquad \frac{}{A \,\&\, B \longrightarrow A}\ \&\textsc{d}_1 \qquad \frac{}{A \,\&\, B \longrightarrow B}\ \&\textsc{d}_2$$

$$\frac{}{\Omega \vdash \top}\ \top\textsc{r} \qquad \text{(no } \top\textsc{d} \text{ rule)}$$

$$\frac{A\, \Omega \vdash B}{\Omega \vdash A \setminus B}\ \setminus\textsc{r} \qquad \frac{}{A\, (A \setminus B) \longrightarrow B}\ \setminus\textsc{d}$$

$$\frac{\Omega\, A \vdash B}{\Omega \vdash B \,/\, A}\ /\textsc{r} \qquad \frac{}{(B \,/\, A)\, A \longrightarrow B}\ /\textsc{d}$$

$$\frac{\Omega \vdash A}{\Omega \vdash A \oplus B}\ \oplus\textsc{r}_1 \qquad \frac{\Omega \vdash B}{\Omega \vdash A \oplus B}\ \oplus\textsc{r}_2 \qquad \frac{\Omega'_L\, A\, \Omega'_R \vdash C \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\, (A \oplus B)\, \Omega'_R \vdash C}\ \oplus\textsc{l}$$

$$\text{(no } \mathbf{0}\textsc{r} \text{ rule)} \qquad \frac{}{\Omega'_L\, \mathbf{0}\, \Omega'_R \vdash C}\ \mathbf{0}\textsc{l}$$

Figure 5.4: A refactoring of the ordered sequent calculus to emphasize that most left rules amount to resource decomposition

$$\frac{}{A \bullet B \longrightarrow A\,B}\ \bullet\text{D} \qquad \frac{}{\mathbf{1} \longrightarrow \cdot}\ \mathbf{1}\text{D}$$

$$\frac{}{A \,\&\, B \longrightarrow A}\ \&\text{D}_1 \qquad \frac{}{A \,\&\, B \longrightarrow B}\ \&\text{D}_2 \qquad \text{(no } \top\text{D rule)}$$

$$\frac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\text{D} \qquad \frac{}{(B \,/\, A)\,A \longrightarrow B}\ /\text{D} \qquad \text{(no } \oplus\text{D and } \mathbf{0}\text{D rules)}$$

$$\frac{\Omega \longrightarrow \Omega'}{\Omega_L\,\Omega\,\Omega_R \longrightarrow \Omega_L\,\Omega'\,\Omega_R}\ \longrightarrow\text{C}$$

$$\frac{}{\Omega \Longrightarrow \Omega}\ \Longrightarrow\text{R} \qquad \frac{\Omega \longrightarrow \Omega' \quad \Omega' \Longrightarrow \Omega''}{\Omega \Longrightarrow \Omega''}\ \Longrightarrow\text{T}$$

Figure 5.5: The rewriting fragment of ordered logic, based on resource decomposition

THEOREM 5.1 (Soundness and completeness). *$\Omega \vdash A$ is derivable in the refactored calculus of fig. 5.4 if, and only if $\Omega \vdash A$ is derivable in the usual ordered sequent calculus (fig. 3.2).*

Proof. Soundness, the right-to-left direction, can be proved by structural induction on the given derivation. The key lemma is the admissibility of $\text{CUT}^{\longrightarrow}$ in the usual ordered sequent calculus:

If $\Omega \longrightarrow \Omega'$ and $\Omega'_L\,\Omega'\,\Omega'_R \vdash C$, then $\Omega'_L\,\Omega\,\Omega'_R \vdash C$.

This lemma can be proved by case analysis of the decomposition $\Omega \longrightarrow \Omega'$, reconstituting the corresponding left rule along the lines of the sketches from figs. 5.1 and 5.3.

Completeness, the left-to-right direction, can be proved by structural induction on the given derivation. The critical cases are the left rules; they are resolved along the lines of the sketches shown in figs. 5.1 and 5.3. □

5.1.2 *An ordered rewriting framework*

Thus far, we have used the decomposition judgment, $\Omega \longrightarrow \Omega'$, and its rules as the basis for a reconfigured sequent-like calculus for ordered logic. Alternatively, we can view decomposition as the foundation of a rewriting system grounded in ordered logic. For example, the decomposition of resource $A \bullet B$ into $A\,B$ by the $\bullet$D rule can also be seen as *rewriting* $A \bullet B$ into $A\,B$. More generally, the decomposition judgment $\Omega \longrightarrow \Omega'$ can be read as "Ω rewrites to Ω'", where contexts Ω are states that are subject to rewriting.

Figure 5.5 summarizes the ordered rewriting system that we obtain from the refactored sequent-like calculus of fig. 5.4. Essentially, the ordered rewriting framework is obtained by discarding all rules except for the decomposition rules. However, if only the decomposition rules are used, rewritings cannot occur within a larger context. For example, the $\setminus$D decomposition rule derives $A\,(A \setminus B) \longrightarrow B$, but $\Omega'_L\,A\,(A \setminus B)\,\Omega'_R \longrightarrow \Omega'_L\,B\,\Omega'_R$ would not be

derivable in general. In the refactored calculus of fig. 5.4, this kind of framing is taken care of by the cut principle for decomposition, $\textsc{cut}^{\longrightarrow}$. To express framing at the level of the $\Omega \longrightarrow \Omega'$ judgment itself, we ensure that rewriting is compatible with concatenation of ordered contexts:

$$\frac{\Omega \longrightarrow \Omega'}{\Omega_L\ \Omega\ \Omega_R \longrightarrow \Omega_L\ \Omega'\ \Omega_R}\ \longrightarrow_{\mathrm{C}}.$$

This is analogous to the compatibility rule for string rewriting within substrings.

Also like in string rewriting, we can form the reflexive, transitive closure of $\longrightarrow$, as a multi-step rewriting relation; we again choose to write the reflexive, transitive closure as $\Longrightarrow$. We prefer the list-like formulation of $\Longrightarrow$ shown in fig. 5.5 because it tends to streamline proofs by structural induction, but, on the basis of the following lemma, we allow ourselves to freely switch to a tree-like formulation as needed.

Lemma 5.2 (Transitivity of $\Longrightarrow$). *If* $\Omega \Longrightarrow \Omega'$ *and* $\Omega' \Longrightarrow \Omega''$, *then* $\Omega \Longrightarrow \Omega''$.

Proof. By induction on the structure of the first trace, $\Omega \Longrightarrow \Omega'$. □

A few remarks about these rewriting relations are in order. First, interpreting the resource decomposition rules as rewriting only confirms our preference for the nullary $\backslash\mathrm{D}$ and $/\mathrm{D}$ rules (eq. 5.2). The $\backslash\mathrm{D}'$ and $/\mathrm{D}'$ rules (eq. 5.1), with their $\Omega \vdash A$ premises, would be problematic as rewriting rules because they would introduce a dependence of rewriting upon general provability and the accompanying proof search would take ordered rewriting too far afield from traditional, syntactic notions of string and multiset rewriting.

Second, multi-step rewriting, $\Longrightarrow$, is incomplete with respect to the usual ordered sequent calculus (fig. 3.2) because all right rules have been discarded.

False claim 5.3 (Completeness). *If* $\Omega \vdash A$, *then* $\Omega \Longrightarrow A$.

Counterexample. The sequent $A \setminus (C \,/\, B) \vdash (A \setminus C) \,/\, B$ is provable, and yet $A \setminus (C \,/\, B) \not\Longrightarrow (A \setminus C) \,/\, B$ (even though $A\ (A \setminus (C \,/\, B))\ B \Longrightarrow C$ does hold). □

As expected from the way in which it was developed, ordered rewriting is, however, sound. To state and prove soundness, we must first define an operation $\bullet\Omega$ that reifies an ordered context as a single proposition (see adjacent figure).

$$\begin{aligned} \bullet(\Omega_1\ \Omega_2) &= (\bullet\Omega_1) \bullet (\bullet\Omega_2) \\ \bullet(\cdot) &= \mathbf{1} \\ \bullet A &= A \end{aligned}$$

Figure 5.6: From ordered contexts to propositions

Lemma 5.4. *For all* Ω *and* C, *if* $\Omega \vdash C$, *then* $\bullet\Omega \vdash C$. *Also,* $\Omega \vdash \bullet\Omega$ *for all* Ω.

Proof. By induction on the structure of the given context, Ω. □

Theorem 5.5 (Soundness). *If* $\Omega \longrightarrow \Omega'$, *then* $\Omega \vdash \bullet\Omega'$. *Also, if* $\Omega \Longrightarrow \Omega'$, *then* $\Omega \vdash \bullet\Omega'$.

Proof. By induction on the structure of the given step or trace. □

5.1.3 *Properties of the ordered rewriting framework*

As we did for the string rewriting framework, we can evaluate the above ordered rewriting framework for confluence, termination, and, most importantly, concurrency.

CONCURRENCY Like string rewriting, ordered rewriting admits concurrency. As an example of concurrent ordered rewriting, observe that, as shown in the adjacent figure, two sequences witness $a\,(a \setminus b)\,(c \,/\, a)\,a \Longrightarrow b\,c$: either the initial state's left half, $a\,(a \setminus b)$, is first rewritten to b and then its right half, $(c \,/\, a)\,a$, is rewritten to c; or *vice versa*, the right half is first rewritten to c and then the left half is rewritten to b.

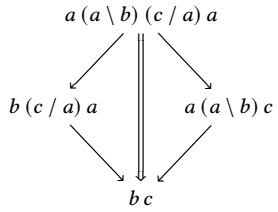


Figure 5.7: An example of concurrency in ordered rewriting

These two sequences differ only in how non-overlapping, and therefore independent, rewritings of the initial state's two halves are interleaved. And so, just as in string rewriting, the two sequences can be – and indeed should be – considered essentially equivalent. The details of how the small-step rewrites are interleaved are irrelevant, so that conceptually, at least, only the big-step trace from $a\,(a \setminus b)\,(c \,/\, a)\,a$ to $b\,c$ remains.

NON-CONFLUENCE Although it admits concurrency, ordered rewriting is not confluent. Confluence would require that all states with a common ancestor, *i.e.*, states Ω_1' and Ω_2' such that $\Omega_1' \Longleftrightarrow \Omega_2'$, be joinable, *i.e.*, $\Omega_1' \Longrightarrow\!\Longleftarrow \Omega_2'$.

FALSE CLAIM 5.6 (Confluence). *If* $\Omega_1' \Longleftrightarrow \Omega_2'$, *then* $\Omega_1' \Longrightarrow\!\Longleftarrow \Omega_2'$.

Counterexamples. Consider $a \,\&\, b$. By the rewriting rules for additive conjunction, $a \longleftarrow a \,\&\, b \longrightarrow b$, and hence $a \Longleftarrow a \,\&\, b \Longrightarrow b$. However, being atoms, neither a nor b reduces. And $a \neq b$, so $a \Longrightarrow\!\Longleftarrow b$ does *not* hold. Even in the $\&$-free fragment, ordered rewriting is not confluent: for example,

$$\leftarrow\!\!\!\!\!\!\cdot\;\; c\,(a \setminus b) \Longleftarrow (c \,/\, a)\,a\,(a \setminus b) \Longrightarrow (c \,/\, a)\,b \;\;\cdot\!\!\!\!\!\!\rightarrow\;. \qquad \square$$

TERMINATION Notice that every rewriting step, $\Omega \longrightarrow \Omega'$, strictly decreases the number of logical connectives that occur in the ordered context. More formally, let $|\Omega|_\star$ be a measure of the number of logical connectives that occur in Ω, as defined in the adjacent figure. We may then prove the following lemma.

$$\begin{aligned} |\Omega_1\;\Omega_2|_\star &= |\Omega_1|_\star + |\Omega_2|_\star \\ |\cdot|_\star &= 0 \\ |A \star B|_\star &= 1 + |A|_\star + |B|_\star \\ &\qquad \text{if } \star = \bullet, \,\&, \setminus, /, \text{ or } \oplus \\ |A|_\star &= 1 \quad \text{if } A = a, \mathbf{1}, \top, \text{ or } \mathbf{0} \end{aligned}$$
Figure 5.8: A measure of the number of logical connectives within an ordered context

LEMMA 5.7. *If* $\Omega \longrightarrow \Omega'$, *then* $|\Omega|_\star > |\Omega'|_\star$. *If* $\Omega \Longrightarrow \Omega'$, *then* $|\Omega|_\star \geq |\Omega'|_\star$.

Proof. By induction on the structure of the rewriting step. $\square$

On the basis of this lemma, we will frequently refer to the rewriting relation, $\longrightarrow$, as the *reduction relation*. We may use this lemma to prove that ordered rewriting is terminating.

THEOREM 5.8 (Termination). *For all ordered contexts* Ω, *every rewriting sequence from* Ω *is finite.*

Proof. Let Ω be an arbitrary ordered context. Beginning from state $\Omega_0 = \Omega$, some state Ω_i will eventually be reached such that either: $\Omega_i \not\longrightarrow$; or $|\Omega_i|_\star = 0$ and $\Omega_i \longrightarrow \Omega_{i+1}$. In the latter case, lemma 5.7 establishes $|\Omega_{i+1}|_\star < 0$, which is impossible because $|-|_\star$ is a measure and hence nonnegative. □

5.2 *A* focused *ordered rewriting framework*

The above ordered rewriting framework is based upon decomposition rules that are very fine-grained. Each step of rewriting decomposes a proposition into only its immediate subformulas, and no further, such as in the very fine-grained step $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow a\,(a \backslash c \bullet a)$. It is not possible to rewrite $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big)$ into $c\,a$ (or even $c \bullet a$) in a single step, although it is possible in several steps:

$$a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow a\,(a \backslash c \bullet a) \longrightarrow c \bullet a \longrightarrow c\,a\,.$$

The decomposition rules are so fine-grained that rewriting may even get stuck in undesirable and unintended ways. For instance, in the previous example, we might have instead nondeterministically committed to rewriting $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big)$ into $a\,(b \backslash \mathbf{1})$ as the first step, and then $a\,(b \backslash \mathbf{1})$ is stuck, with no further rewritings possible:

$$a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow a\,(b \backslash \mathbf{1}) \not\longrightarrow .$$

Instead, we would rather have a coarser notion of decomposition so that $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow c\,a$ is a single step[6] and, conversely, so that $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow \Omega'$ only if $\Omega' = c\,a$.

[6] Or so that $a\,\big((a \backslash c \bullet a) \mathbin{\&} (b \backslash \mathbf{1})\big) \longrightarrow c \bullet a$, at least, is a single step.

Focusing, as developed by Andreoli,[7] provides just the right coarse-grained decomposition through its complementary inversion and chaining strategies for proof search. An inversion phase groups together successive invertible rules, and a chaining phase groups together successive noninvertible rules that are applied to a single *in-focus* proposition; together, a chaining phase followed by an inversion phase constitutes a *bipole*. Rather than having each of these rules give rise to a separate step, we can treat the entire bipole as an atomic step of rewriting.

[7] Andreoli 1992.

The rewriting framework described above might be termed *unfocused* ordered rewriting; we will now describe a focused ordered rewriting framework.

This idea of using focusing to increase the granularity of rewriting steps dates back to, at least, the Concurrent Logical Framework (CLF)[8] and was later streamlined in linear logic by Cervesato and Scedrov[9] Simmons[10] has studied a focused ordered rewriting framework, though in a somewhat different formulation than the one we present here.

[8] Watkins et al. 2002.

[9] Cervesato and Scedrov 2009.

[10] Simmons 2012.

The ordered propositions are now polarized into positive and negative classes, or *polarities*,[11] according to the invertibility of their sequent calculus

[11] Andreoli 1992.

rules; two 'shift' operators, $\downarrow$ and $\uparrow$, mediate between the two classes.

$$A^+ ::= a^+ \mid A^+ \bullet B^+ \mid \mathbf{1} \mid \downarrow A^-$$
$$A^- ::= A^+ \setminus B^- \mid B^- / A^+ \mid A^- \mathbin{\&} B^- \mid \top \mid \uparrow A^+$$

The positive propositions, A^+, are those propositions that have invertible left rules, such as ordered conjunction; the negative propositions, A^-, are those that have invertible right rules, such as the left- and right-handed implications. For reasons that will become clear in chapter 6, we choose to assign a positive polarity to all atomic propositions, a^+.

To streamline the syntax, we will often make two elisions. Because all atomic propositions have positive polarity, we will often write a instead of a^+. We will also often elide any $\downarrow$ and $\uparrow$ shifts that are necessitated by the polarities of the remaining connectives; we call such shifts the minimal shifts. For example, we might write $a \bullet (b \setminus \mathbf{1})$ in place of the more verbose, implied $a^+ \bullet \downarrow(b^+ \setminus \uparrow\mathbf{1})$.

Ordered contexts are formed as the free monoid over negative propositions and positive atoms:

$$\Omega, \Delta ::= \Omega_1\ \Omega_2 \mid \cdot \mid A^- \mid a^+ .$$

As usual, we do not distinguish those ordered contexts that are equivalent up to the monoid laws. Notice that we use the same metavariables for these polarized ordered contexts as we did for the unpolarized contexts of the preceding (unfocused) ordered rewriting framework – the intended context, either polarized or not, will always be clear from context.

We may also reify an ordered context Ω as a positive proposition, $\bullet\Omega$, using the operation defined in the neighboring figure. This operation is the polarized analogue of the operation shown in fig. 5.6.

$$\bullet(\Omega_1\ \Omega_2) = (\bullet\Omega_1) \bullet (\bullet\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$\bullet A^- = \downarrow A^-$$
$$\bullet a^+ = a^+$$

Figure 5.9: Reifying an ordered context as a positive proposition

Each class of propositions is then equipped with its own focusing judgment: a *left-focus judgment*, $\Omega_L\ [A^-]\ \Omega_R \Vdash C^+$, that focuses on a negative proposition, A^-, that occurs to the left of the turnstile; and a *right-focus judgment*, $[A^+] \dashv\!\shortmid\ \Omega$, that focuses on a positive proposition, A^+, that occurs to the right of the turnstile.[12]

Following Zeilberger,[13] each of these judgments can be read as a function that provides a form of extended decomposition – the in-focus proposition is decomposed beyond its immediate subformulas, until subformulas of the opposite polarity are reached. The two focusing judgments are defined inductively on the structure of the in-focus proposition, with the left-focus judgment depending on the right-focus judgment (though not *vice versa*).

The right-focus judgment, $[A^+] \dashv\!\shortmid\ \Omega$, decomposes A^+ into the ordered context Ω of its nearest negative subformulas, treating A^+ as input and Ω as

[12] We choose a left-facing turnstile for the right-focus judgment to emphasize its input/output mode; see the next paragraph.

[13] Zeilberger 2008.

output. The judgment is given by the following rules.

$$\frac{[A^+] \dashv\Vdash \Omega_1 \quad [B^+] \dashv\Vdash \Omega_2}{[A^+ \bullet B^+] \dashv\Vdash \Omega_1\ \Omega_2}\ \bullet\text{R} \qquad \frac{}{[\mathbf{1}] \dashv\Vdash \cdot}\ \mathbf{1}\text{R}$$

$$\frac{}{[a^+] \dashv\Vdash a^+}\ \text{ID}^{a^+} \qquad \frac{}{[\downarrow A^-] \dashv\Vdash A^-}\ \downarrow\text{R}$$

Ordered conjunctions $A^+ \bullet B^+$ are decomposed into $\Omega_1\ \Omega_2$ by inductively decomposing A^+ and B^+ into Ω_1 and Ω_2, respectively, and $\mathbf{1}$ is decomposed into the empty context. Atoms a^+ are not decomposed further[14], and $\downarrow A^-$ is decomposed into its immediate subformula of negative polarity, A^-.

[14] Alternatively, following Simmons (2012), atoms a^+ could be decomposed to suspensions $\langle a^+ \rangle$, but we choose not to do that here.

This right-focus judgment is a left inverse of the $\bullet(-)$ operation:

Lemma 5.9. $[\bullet\Omega] \dashv\Vdash \Omega'$ *if, and only if,* $\Omega = \Omega'$.

Proof. Each direction is separately proved by structural induction on the context Ω. □

The left-focus judgment, $\Omega_L\ [A^-]\ \Omega_R \Vdash C^+$, decomposes A^- into the ordered contexts Ω_L and Ω_R and positive subformula C^+, treating A^- as input and Ω_L, Ω_R, and C^+ as outputs. The judgment is given by the following rules.

$$\frac{[A^+] \dashv\Vdash \Omega_A \quad \Omega_L\ [B^-]\ \Omega_R \Vdash C^+}{\Omega_L\ \Omega_A\ [A^+ \setminus B^-]\ \Omega_R \Vdash C^+}\ \setminus\text{L} \qquad \frac{[A^+] \dashv\Vdash \Omega_A \quad \Omega_L\ [B^-]\ \Omega_R \Vdash C^+}{\Omega_L\ [B^- \,/\, A^+]\ \Omega_A\ \Omega_R \Vdash C^+}\ /\text{L}$$

$$\frac{\Omega_L\ [A^-]\ \Omega_R \Vdash C^+}{\Omega_L\ [A^- \,\&\, B^-]\ \Omega_R \Vdash C^+}\ \&\text{L}_1 \qquad \frac{\Omega_L\ [B^-]\ \Omega_R \Vdash C^+}{\Omega_L\ [A^- \,\&\, B^-]\ \Omega_R \Vdash C^+}\ \&\text{L}_2 \qquad \text{(no } \top\text{L rule)}$$

$$\frac{}{[\uparrow A^+] \Vdash A^+}\ \uparrow\text{L}$$

The left-focus judgment's rules parallel the usual sequent calculus rules, but maintain focus on the immediate subformulas – left focus for subformulas of negative polarity and right focus for subformulas of positive polarity. The $\uparrow$L rule ends left focus by decomposing an $\uparrow A^+$ antecedent into an A^+ consequent.

Unlike the right-focus judgment, the left-focus judgment describes a relation (or nondeterministic function), owing to the two rules, $\&\text{L}_1$ and $\&\text{L}_2$, that may apply to alternative conjunctions. For example, the following are derivable.

$$a^+\ [(a^+ \setminus \uparrow(c^+ \bullet a^+)) \,\&\, (b^+ \setminus \uparrow\mathbf{1})] \Vdash c^+ \bullet a^+$$
$$b^+\ [(a^+ \setminus \uparrow(c^+ \bullet a^+)) \,\&\, (b^+ \setminus \uparrow\mathbf{1})] \Vdash \mathbf{1}$$

A FOCUSED REWRITING STEP ARISES when a negative proposition, A^-, is put into focus and the resulting consequent, C^+, is subsequently decomposed into the ordered context.[15] In addition, the compatibility rule $\longrightarrow$C is retained.

[15] Writing $[B^+] \dashv\Vdash \Omega'$ for the right-focus judgment gives this rule the flavor of a cut principle.

$$\frac{\Omega_L\ [A^-]\ \Omega_R \Vdash C^+ \quad [C^+] \dashv\Vdash \Omega'}{\Omega_L\ A^-\ \Omega_R \longrightarrow \Omega'}\ \longrightarrow\text{I} \qquad \frac{\Omega \longrightarrow \Omega'}{\Omega_L\ \Omega\ \Omega_R \longrightarrow \Omega_L\ \Omega'\ \Omega_R}\ \longrightarrow\text{C}$$

Figure 5.10: A framework for focused ordered rewriting

$$\text{POSITIVE PROPS.} \quad A^+ ::= A^+ \bullet B^+ \mid \mathbf{1} \mid a^+ \mid {\downarrow}A^-$$

$$\text{NEGATIVE PROPS.} \quad A^- ::= A^+ \setminus B^- \mid B^- / A^+ \mid A^- \mathbin{\&} B^- \mid \top \mid \hat{p}^- \mid {\uparrow}A^+$$

$$\text{CONTEXTS} \quad \Omega ::= \Omega_1\, \Omega_2 \mid \cdot \mid A^- \mid a^+$$

$$\frac{[A^+] \dashv\Vdash \Omega_1 \quad [B^+] \dashv\Vdash \Omega_2}{[A^+ \bullet B^+] \dashv\Vdash \Omega_1\, \Omega_2}\ \bullet\text{R} \qquad \frac{}{[\mathbf{1}] \dashv\Vdash \cdot}\ \mathbf{1}\text{R}$$

$$\frac{}{[a^+] \dashv\Vdash a^+}\ \text{ID}^{a^+} \qquad \frac{}{[{\downarrow}A^-] \dashv\Vdash A^-}\ {\downarrow}\text{R}$$

$$\frac{[A^+] \dashv\Vdash \Omega_A \quad \Omega_L\, [B^-]\, \Omega_R \Vdash C^+}{\Omega_L\, \Omega_A\, [A^+ \setminus B^-]\, \Omega_R \Vdash C^+}\ \setminus\text{L} \qquad \frac{[A^+] \dashv\Vdash \Omega_A \quad \Omega_L\, [B^-]\, \Omega_R \Vdash C^+}{\Omega_L\, [B^- / A^+]\, \Omega_A\, \Omega_R \Vdash C^+}\ /\text{L}$$

$$\frac{\Omega_L\, [A^-]\, \Omega_R \Vdash C^+}{\Omega_L\, [A^- \mathbin{\&} B^-]\, \Omega_R \Vdash C^+}\ \&\text{L}_1 \qquad \frac{\Omega_L\, [B^-]\, \Omega_R \Vdash C^+}{\Omega_L\, [A^- \mathbin{\&} B^-]\, \Omega_R \Vdash C^+}\ \&\text{L}_2 \qquad \text{(no } \top\text{L rule)}$$

$$\frac{}{[{\uparrow}A^+] \Vdash A^+}\ {\uparrow}\text{L}$$

$$\frac{\Omega_L\, [A^-]\, \Omega_R \Vdash C^+ \quad [C^+] \dashv\Vdash \Omega'}{\Omega_L\, A^-\, \Omega_R \longrightarrow \Omega'}\ \longrightarrow\text{I} \qquad \frac{\Omega \longrightarrow \Omega'}{\Omega_L\, \Omega\, \Omega_R \longrightarrow \Omega_L\, \Omega'\, \Omega_R}\ \longrightarrow\text{C}$$

$$\frac{}{\Omega \Longrightarrow \Omega}\ \Longrightarrow\text{R} \qquad \frac{\Omega \longrightarrow \Omega' \quad \Omega' \Longrightarrow \Omega''}{\Omega \Longrightarrow \Omega''}\ \Longrightarrow\text{T}$$

With this $\longrightarrow$I rule, it is indeed possible to rewrite

$$a^+ \left((a^+ \setminus {\uparrow}(c^+ \bullet a^+)) \mathbin{\&} (b^+ \setminus {\uparrow}\mathbf{1})\right) \longrightarrow c^+\, a^+$$

in a single, atomic step because both $a^+\, [(a^+ \setminus {\uparrow}(c^+ \bullet a^+)) \mathbin{\&} (b^+ \setminus {\uparrow}\mathbf{1})] \Vdash c^+ \bullet a^+$ and $[c^+ \bullet a^+] \dashv\Vdash c^+\, a^+$ are derivable. Moreover, the larger granularity afforded by the left- and right-focus judgments precludes the small steps that led to unintended stuck states. For example:

$$a^+ \left((a^+ \setminus {\uparrow}(c^+ \bullet a^+)) \mathbin{\&} (b^+ \setminus {\uparrow}\mathbf{1})\right) \longrightarrow \Omega' \ \text{ only if } \ \Omega' = c^+\, a^+\,.$$

and so

$$a^+ \left((a^+ \setminus {\uparrow}(c^+ \bullet a^+)) \mathbin{\&} (b^+ \setminus {\uparrow}\mathbf{1})\right) \not\longrightarrow a^+\, (b^+ \setminus {\uparrow}\mathbf{1})\,.$$

5.3 *Using shifts to control focusing*

With careful placement of shifts, it is possible to control the behavior of focused rewriting. It is even possible to embed the unfocused ordered rewriting

framework of section 5.1.2 and a weakly focused ordered rewriting framework within focused ordered rewriting in an operationally faithful way, as we now show.

5.3.1 *Embedding unfocused ordered rewriting*

With careful placement of additional, non-minimal shifts, it is possible to embed unfocused ordered rewriting within the focused ordered rewriting framework in a operationally faithful way. Specifically, we can define a mapping, $(-)^\blacksquare$, from contexts of unpolarized propositions to contexts of negative propositions and positive atoms in a way that strongly respects the operational behavior of unfocused ordered rewriting:

- $\Omega \longrightarrow \Omega'$ implies $\Omega^\blacksquare \longrightarrow \Omega'^\blacksquare$; and
- $\Omega^\blacksquare \longrightarrow \Delta'$ implies $\Omega \longrightarrow \Omega'$, for some Ω' such that $\Delta' = \Omega'^\blacksquare$.

Because $(-)^\blacksquare$ is a total function, these properties thus establish it as a *strong reduction bisimulation.*[16]

[16] Sangiorgi and Walker 2003.

Essentially, this embedding inserts a double shift, $\downarrow\uparrow$, in front of each proper, nonatomic subformula. These double shifts cause chaining and inversion to be interrupted after each step, forcing the focused rewriting to mimic the small-step behavior of unfocused rewriting.

More specifically, $A^\blacksquare$ prepends an $\uparrow$ shift whenever the top-level connective of A has positive polarity. Consequently, $A^\blacksquare$ is either: a positive atom, exactly when A is atomic; or otherwise a negative proposition. Also, the mapping $(-)^\blacksquare$ relies on the $\bullet$ operation on contexts – thus, $\bullet A^\blacksquare$ inserts a $\downarrow$ shift in front of $A^\blacksquare$ exactly when $A^\blacksquare$ and hence A are nonatomic. Together, these features serve to insert a double shift, $\downarrow\uparrow$, in front of each proper, nonatomic subformula of A.

$$
\begin{aligned}
(\Omega_1\ \Omega_2)^\blacksquare &= \Omega_1^\blacksquare\ \Omega_2^\blacksquare \\
(\cdot)^\blacksquare &= \cdot \\
a^\blacksquare &= a^+ \\
(A \bullet B)^\blacksquare &= \uparrow((\bullet A^\blacksquare) \bullet (\bullet B^\blacksquare)) \\
\mathbf{1}^\blacksquare &= \uparrow\mathbf{1} \\
(A \setminus B)^\blacksquare &= (\bullet A^\blacksquare) \setminus \uparrow(\bullet B^\blacksquare) \\
(B \,/\, A)^\blacksquare &= \uparrow(\bullet B^\blacksquare) \,/\, (\bullet A^\blacksquare) \\
(A \mathbin{\&} B)^\blacksquare &= \uparrow(\bullet A^\blacksquare) \mathbin{\&} \uparrow(\bullet B^\blacksquare) \\
\top^\blacksquare &= \top
\end{aligned}
$$

Figure 5.11: An embedding of unfocused ordered rewriting within focused ordered rewriting

Theorem 5.10. *The embedding $(-)^\blacksquare$ satisfies the following properties.*

- *If $\Omega \longrightarrow \Omega'$, then $\Omega^\blacksquare \longrightarrow \Omega'^\blacksquare$.*
- *If $\Omega^\blacksquare = \Delta \longrightarrow \Delta'$, then $\Omega \longrightarrow \Omega'$ for some Ω' such that $\Delta' = \Omega'^\blacksquare$.*

Proof. The proofs of these properties require a straightforward lemma: for all unpolarized propositions A,

$$[\bullet A^\blacksquare] \dashv \Delta \text{ if, and only if, } \Delta = A^\blacksquare.$$

The first property is then proved by induction over the structure of the given rewriting step, $\Omega \longrightarrow \Omega'$. As an example, consider the case in which $\Omega = A\ (A \setminus B) \longrightarrow B = \Omega'$. By definition, $\Omega^\blacksquare = A^\blacksquare\ ((\bullet A^\blacksquare) \setminus \uparrow(\bullet B^\blacksquare))$ and $\Omega'^\blacksquare = B^\blacksquare$; we can indeed derive $A^\blacksquare\ [(\bullet A^\blacksquare) \setminus \uparrow(\bullet B^\blacksquare)] \Vdash \bullet B^\blacksquare$ and $[\bullet B^\blacksquare] \dashv B^\blacksquare$. So, as required, $\Omega^\blacksquare = A^\blacksquare\ ((\bullet A^\blacksquare) \setminus \uparrow(\bullet B^\blacksquare)) \longrightarrow B^\blacksquare = \Omega'^\blacksquare$.

The second property is also proved by induction over the structure of the given rewriting step, this time $\Omega^\blacksquare = \Delta \longrightarrow \Delta'$. As an example, consider the case in which $\Omega_L^\blacksquare\ [(\bullet A^\blacksquare) \setminus \uparrow(\bullet B^\blacksquare)]\ \Omega_R^\blacksquare \Vdash C^+$ and $[C^+] \dashv \Delta'$, for some Ω_L, A, B,

$$\textsc{contexts} \quad \Omega^+ ::= \Omega_1^+\,\Omega_2^+ \mid \cdot \mid A^+$$

$$\frac{\Omega_L^+\,[B^-]\,\Omega_R^+ \Vdash C^+}{\Omega_L^+\,a^+\,[a^+ \setminus B^-]\,\Omega_R^+ \Vdash C^+}\ \setminus\text{L} \qquad \frac{\Omega_L^+\,[B^-]\,\Omega_R^+ \Vdash C^+}{\Omega_L^+\,[B^- \,/\, a^+]\,a^+\,\Omega_R^+ \Vdash C^+}\ /\text{L}$$

$$\frac{\Omega_L^+\,[A^-]\,\Omega_R^+ \Vdash C^+}{\Omega_L^+\,[A^- \,\&\, B^-]\,\Omega_R^+ \Vdash C^+}\ \&\text{L}_1 \qquad \frac{\Omega_L^+\,[B^-]\,\Omega_R^+ \Vdash C^+}{\Omega_L^+\,[A^- \,\&\, B^-]\,\Omega_R^+ \Vdash C^+}\ \&\text{L}_2 \qquad \text{(no } \top\text{L rule)}$$

$$\frac{}{[\uparrow A^+] \Vdash A^+}\ \uparrow\text{L}$$

$$\frac{\Omega_L^+\,[A^-]\,\Omega_R^+ \Vdash C^+}{\Omega_L^+ \downarrow A^-\,\Omega_R^+ \longrightarrow C^+}\ \downarrow\text{D} \qquad \frac{}{A^+ \bullet B^+ \longrightarrow A^+\,B^+}\ \bullet\text{D} \qquad \frac{}{\mathbf{1} \longrightarrow \cdot}\ \mathbf{1}\text{D}$$

$$\frac{\Omega^+ \longrightarrow \Omega'^+}{\Omega_L^+\,\Omega^+\,\Omega_R^+ \longrightarrow \Omega_L^+\,\Omega'^+\,\Omega_R^+}\ \longrightarrow\text{C}$$

Figure 5.12: A framework for *weakly* focused ordered rewriting

Ω_R, and C^+ such that $\Omega = \Omega_L\ (A \setminus B)\ \Omega_R$. By inversion and the aforementioned lemma, we have $\Omega_L = A$, $\Omega_R = (\cdot)$, $C^+ = \bullet B^\blacksquare$, and $\Delta' = B^\blacksquare$. Indeed, as required, $\Omega = A\ (A \setminus B) \longrightarrow B = \Omega'$ and $\Delta' = \Omega'^\blacksquare$. □

5.3.2 *Embedding weakly focused ordered rewriting*

It is similarly possible to embed *weakly* focused ordered rewriting, a rewriting discipline based on weak focusing[17] in which the granularity of steps lies between that of the unfocused and (strongly[18]) focused ordered rewriting frameworks. More specifically, weak focusing differs from (strong) focusing in that it retains chaining but abandons eager inversion. For example, with weakly focused rewriting,

$$a^+ \downarrow\big((a^+ \setminus \uparrow(c^+ \bullet a^+)) \,\&\, (b^+ \setminus \uparrow\mathbf{1})\big) \longrightarrow c^+ \bullet a^+ \longrightarrow c^+\,a^+\,,$$

where the inversion of $c^+ \bullet a^+$ is now an atomic step of its own.

This weakly focused rewriting discipline could be achieved as an independent system with the rules shown in fig. 5.12. Notice that weakly focused rewriting restricts the left- and right-handed implications to have only atomic premises. Although weak focusing is well-defined for arbitrary implications, it is not clear how to give a *rewriting* interpretation of weak focusing unless this restriction is made.

In fact, there is a better approach than using weakly focused ordered rewriting as yet another independent rewriting system. Instead of using weakly focused rewriting directly, we can embed it within (strongly) focused ordered rewriting by inserting shifts at specific locations and then use that embedding. From here on, we will exclusively use this embedding when weakly focused ordered rewriting is needed (which is only in chapter 10).

[17] Laurent 2002; Simmons and Pfenning 2011b.

[18] Commonly known as fully focused.

$$\begin{aligned}
(\Omega_1^+\,\Omega_2^+)^\square &= (\Omega_1^+)^\square\,(\Omega_2^+)^\square \\
(\cdot)^\square &= \cdot \\
(A^+)^\square &= (A^+)^\boxplus \\
\hline
(a^+)^\boxplus &= a^+ \\
(A^+ \bullet B^+)^\boxplus &= \uparrow((\bullet(A^+)^\boxplus) \bullet (\bullet(B^+)^\boxplus)) \\
\mathbf{1}^\boxplus &= \uparrow\mathbf{1} \\
(\downarrow A^-)^\boxplus &= (A^-)^\boxminus \\
\hline
(a^+ \setminus B^-)^\boxminus &= a^+ \setminus (B^-)^\boxminus \\
(B^- \,/\, a^+)^\boxminus &= (B^-)^\boxminus \,/\, a^+ \\
(A^- \,\&\, B^-)^\boxminus &= (A^-)^\boxminus \,\&\, (B^-)^\boxminus \\
\top^\boxminus &= \top \\
(\uparrow A^+)^\boxminus &= \uparrow(\bullet(A^+)^\boxplus)
\end{aligned}$$

Figure 5.13: An embedding of weakly focused ordered rewriting within (strongly) focused ordered rewriting

Theorem 5.11. *The embedding* $(-)^{\square}$ *satisfies the following properties.*

- *If* $\Omega^+ \longrightarrow \Omega'^+$, *then* $(\Omega^+)^{\square} \longrightarrow (\Omega'^+)^{\square}$.
- *If* $(\Omega^+)^{\square} \longrightarrow \Delta'$, *then* $\Omega^+ \longrightarrow \Omega'^+$ *for some* Ω'^+ *such that* $\Delta' = (\Omega'^+)^{\square}$.

Proof. The proofs of these properties require two relatively straightforward lemmas: for all polarized propositions A^+ and A^-,

- $[\bullet(A^+)^{\boxplus}] \dashv \Delta$ if, and only if, $\Delta = (A^+)^{\square}$; and
- $\Delta_L\ [(A^-)^{\boxminus}]\ \Delta_R \Vdash B^+$ if, and only if, $\Omega_L^+\ [A^-]\ \Omega_R^+ \Vdash C^+$ and $\Delta_L = (\Omega_L^+)^{\square}$, $\Delta_R = (\Omega_R^+)^{\square}$, and $B^+ = \bullet(C^+)^{\boxplus}$.

Both lemmas are proved by structural induction on the polarized proposition, A^+ and A^-, respectively.

The first of the above properties is then proved by induction over the structure of the given weakly focused rewriting step, $\Omega^+ \longrightarrow \Omega'^+$. As an example, consider the case in which $\Omega_L^+ \downarrow A^-\ \Omega_R^+ \longrightarrow C^+$ because $\Omega_L^+\ [A^-]\ \Omega_R^+ \Vdash C^+$. By the above lemmas, $(\Omega_L^+)^{\square}\ [(A^-)^{\boxminus}]\ (\Omega_R^+)^{\square} \Vdash \bullet(C^+)^{\boxplus}$ and $[\bullet(C^+)^{\boxplus}] \dashv (C^+)^{\square}$ hold in the fully focused calculus. And so, $(\Omega_L^+)^{\square}\ (\downarrow A^-)^{\square}\ (\Omega_R^+)^{\square} \longrightarrow (C^+)^{\square}$.

The second property is also proved by induction over the structure of the given rewriting step, this time the fully focused $(\Omega^+)^{\square} \longrightarrow \Delta'$. As an example, consider the case in which $\Delta_L\ [a_1^+ \setminus (A_2^-)^{\boxminus}]\ \Delta_R \Vdash B^+$ and $[B^+] \dashv \Delta'$. Inversion yields $\Delta'_L\ [(A_2^-)^{\boxminus}]\ \Delta_R \Vdash B^+$ for some Δ'_L such that $\Delta_L = \Delta'_L\ a_1^+$. Then, by the above lemma, $\Omega_L^+\ [A_2^-]\ \Omega_R^+ \Vdash C^+$ holds in the weakly focused calculus, with $\Delta'_L = (\Omega_L^+)^{\square}$, $\Delta_R = (\Omega_R^+)^{\square}$, and $B^+ = \bullet(C^+)^{\boxplus}$. Appending the $\setminus\text{L}$ rule, it follows that $\Omega_L^+\ a_1^+\ [a_1^+ \setminus A_2^-]\ \Omega_R^+ \Vdash C^+$, and so $\Omega_L^+\ a_1^+ \downarrow (a_1^+ \setminus A_2^-)\ \Omega_R^+ \longrightarrow C^+$. Also notice that $\Delta_L = (\Omega_L^+\ a_1^+)^{\square}$ and $\Delta' = (C^+)^{\square}$, as required. □

6

Choreographies: A formula-as-process interpretation of ordered rewriting

In chapter 4, we saw that string rewriting can be used to specify the dynamics of concurrent systems, but that those specifications are quite abstract. Even the operational semantics is left completely abstract: String rewriting is a state-transformation model of concurrency, with axioms $w \longrightarrow w'$ stating merely that a substring of the form w may be replaced, en masse, with w'. Nothing is said about how this replacement is achieved – permitted rewritings just *happen*, as if a central, meta-level actor schedules and otherwise coordinates rewriting, with substrings and their constituent symbols as mere passive accessories.

In the previous chapter, we presented a different rewriting framework, derived from the (focused) ordered sequent calculus and closely related to the Lambek calculus.[1] Ordered rewriting, in both its unfocused and focused variants, continues to leave the operational semantics abstract, as if a central, meta-level actor governs rewriting.

[1] Lambek 1958.

The string rewriting and (strongly) focused ordered rewriting frameworks are both expressive enough to specify the dynamics of concurrent systems that have chain topologies[2], and the ordered rewriting framework's logical foundations make it a proof-construction approach to concurrency. But without a concrete operational semantics neither framework is yet suitable for our ultimate goal – to establish the relationship between proof-construction and proof-reduction descriptions of concurrency. In other words, string rewriting and focused ordered rewriting frameworks are well-suited to specification but not to programming.

[2] See sections 4.1.3 and 5.1.3.

This chapter takes three significant steps away from pure specification and toward programming, using the focused ordered rewriting framework of the previous chapter as a stepping stone.

- In section 6.1, we refine the focused ordered rewriting framework of the previous chapter into one that can be given a *formula-as-process* interpretation[3] in which rewriting faithfully represents message-passing commu-

[3] This interpretation is *very* closely related to the process-as-formula view of concurrency put forth by Miller (1992) and Cervesato and Scedrov (2009). For us, however, the logical aspects, and propositions in particular, are conceptually prior to any notion of process, hence our use of the reversed *formula-as-process* terminology.

nication among concurrent processes that are arranged in a linear topology. In this way, the formula-as-process interpretation assigns a concrete operational semantics to ordered rewriting, nudging it away from a state-transformation model of concurrency and toward a process-based model.

Specifically, we show that atomic propositions may be interpreted as messages; the other, non-atomic propositions, as processes; contexts, as configurations comprised of those messages and processes; and rewriting, as message-passing communication among a configuration's constituent processes. Perhaps surprisingly, only three small tweaks to the structure of propositions are needed to make this formula-as-process interpretation viable.

- With this new formula-as-process perspective and its accompanying message-passing semantics, (focused) ordered rewriting can be understood in terms of local interactions alone. By analogy with the π-calculus's operational semantics, the existing rewriting relation, $\longrightarrow$, serves as a reduction semantics, but now an equivalent, labeled transition semantics can also be given (section 6.2).

  The labeled transition semantics describes how ordered contexts, now understood as process configurations, interact with neighboring contexts. It thus goes hand-in-hand with the formula-as-process interpretation in establishing a concrete, local operational semantics for ordered rewriting.

- Having established a formula-as-process refinement of the focused ordered rewriting framework that permits only local, message-passing interactions, we then revisit string rewriting specifications.

  In section 6.3, we describe, first informally and then formally, a method for operationalizing, or *choreographing*, the string rewriting specifications within the formula-as-process ordered rewriting framework. Symbols in the specification's alphabet are uniquely mapped to propositions, thereby casting each symbol in one of two roles – either a message role or a process role.

  Not all such role assignments give rise to well-formed choreographies, however. But, for those that do, the resulting choreography adequately embeds the string rewriting specification: the specification's axioms are in one-to-one correspondence with the choreography's derivable ordered rewritings, as we prove in corollary 6.11. Stated differently, the string rewriting specification and choreography will be (strongly) bisimilar, with the role assignment being a bisimulation that witnesses their bisimilarity.

Even though this chapter introduces a notion of process, it should be noted that computation is still driven by derivability and proof construction, not by proof reduction. Only in part III will we begin to examine a proof-reduction account of concurrency.

6.1 *Refining ordered rewriting: A formula-as-process interpretation*

In this section, we present the formula-as-process interpretation of focused ordered rewriting sketched above.

More specifically, (positive) atoms, a^+, may be viewed as messages, and negative propositions, A^-, as processes that receive and react to those messages. Ordered contexts, Ω, which consist of negative propositions and (positive) atoms, are then chain-topology run-time configurations of processes and messages. And positive propositions, A^+, which reify ordered contexts as propositions, are process expressions that reify configurations. Lastly, but most importantly, the rewriting relation, $\longrightarrow$, is viewed as a reduction semantics for message-passing communication among the processes in a configuration.

Perhaps surprisingly, only three small tweaks to the structure of propositions are needed to make this formula-as-process reading viable.

- The (positive) atoms are now partitioned into two classes, left- and right-directed atoms, to allow us to identify the direction in which a message is flowing.
- The left- and right-handed implications are now restricted to have atomic premises with a complementary direction , so that they may then be cleanly interpreted as input processes that receive individual incoming messages.
- The negative propositions are extended with coinductively defined propositions, $\hat{p}^- \triangleq A^-$, that will correspond to recursive processes.

Together, the first two of these tweaks serve to provide a modicum of static typing for the otherwise untyped processes, as we will discuss in further detail in section 6.1.2.

$\underset{\leftarrow}{a}^+$	*left-directed message*
$\underset{\rightarrow}{a}^+$	*right-directed message*
A^-	*message-passing process*
Ω	*run-time process configuration*
A^+	*configuration reified as an expression*

Table 6.1: A formula-as-process interpretation of polarized ordered propositions and contexts

Positive atoms, as mentioned previously, are now partitioned into two classes, left- and right-directed atoms, to allow us to identify the direction in which a message is flowing. Using a partition, we ensure that each atom has a unique direction that is consistently used across all instances of that atom.

These directions are denoted by an arrow placed below the atom: Left-directed atoms, $\underset{\leftarrow}{a}^+$, are messages that are being sent to the left; right-directed atoms, $\underset{\rightarrow}{a}^+$, are messages that are being sent to the right. We will often elide the polarity annotation, writing $\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{a}$ in place of $\underset{\leftarrow}{a}^+$ and $\underset{\rightarrow}{a}^+$.

Negative propositions, A^-, are processes that receive and react to messages.

$$A^-, B^- ::= \underset{\rightarrow}{a}^+ \setminus B^- \mid B^- / \underset{\leftarrow}{a}^+ \mid A^- \,\&\, B^- \mid \top \mid \uparrow A^+ \mid \hat{p}^-$$

Instead of the more general $A^+ \setminus B^-$ and B^- / A^+, left- and right-handed implications are now restricted to be only $\underset{\rightarrow}{a}^+ \setminus B^-$ and $B^- / \underset{\leftarrow}{a}^+$. These propositions

$\underset{\rightarrow}{a}^+ \setminus B^-$	*receive message $\underset{\rightarrow}{a}^+$ from the right*
$B^- / \underset{\leftarrow}{a}^+$	*receive message $\underset{\leftarrow}{a}^+$ from the left*
$A^- \,\&\, B^-$	*nondeterministic branching*
$\top$	*stuck process*
$\uparrow A^+$	*quoted configuration*
$\hat{p}^-$	*call a recursively defined process*

Table 6.2: A formula-as-process interpretation of negative propositions

are then interpreted as input processes: $\underset{\rightarrow}{a}^+ \setminus B^-$ is a process that waits to receive a message, $\underset{\rightarrow}{a}^+$, from its left-hand neighbor and then continues as B^-; symmetrically, $B^- / \underset{\leftarrow}{a}^+$ is a process that awaits message $\underset{\leftarrow}{a}^+$ from its right-hand neighbor. Because implications are restricted to atoms with *complementary* direction, processes cannot re-capture messages that they just sent to other processes.[4]

[4] See section 6.1.2 for more discussion.

The proposition $A^- \,\&\, B^-$ is interpreted as a process that branches nondeterministically, continuing as either A^- or B^-. And $\top$, as the nullary form of $\&$, is a stuck process that cannot continue. The proposition $\uparrow A^+$ is interpreted as a process that holds a suspended, or quoted, configuration: when the process $\uparrow A^+$ is executed, it unfolds to that configuration. Lastly, $\hat{p}^-$ is a coinductively defined negative proposition that is interpreted as a recursive process, which we discuss in more detail in section 6.1.3.

Ordered contexts, Ω, are interpreted as chain-topology run-time configurations of processes and the messages that pass between them.

$$\Omega, \Delta ::= \Omega_1 \; \Omega_2 \mid \cdot \mid \omega$$
$$\omega ::= A^- \mid \underset{\leftarrow}{a}^+ \mid \underset{\rightarrow}{a}^+$$

$\Omega_1 \; \Omega_2$	*parallel composition of configurations*
$(\cdot)$	*empty configuration*
A^-	*single-process configuration*
$\underset{\leftarrow}{a}^+$	*left-directed message*
$\underset{\rightarrow}{a}^+$	*right-directed message*

Table 6.3: A formula-as-process interpretation of contexts

Just as ordered contexts form a monoid over negative propositions and positive atoms, their formula-as-process interpretation forms a monoid over processes and messages. The monoid operation is now parallel, end-to-end composition of process configurations: $\Omega_1 \; \Omega_2$ composes the configurations Ω_1 and Ω_2 so that they may interact along their mutual interface. The empty context, $(\cdot)$, is now the empty configuration.

As usual, we do not distinguish configurations that are equivalent up to the monoid's associativity and unit laws. This equivalence acts as an implicit structural congruence, of the sort found more explicitly in the π-calculus.

With the introduction of atom directions, it will often be useful to describe *message contexts*, contexts that contain only atoms of one direction or the other. We will use the metavariables $\underset{\leftarrow}{\Omega}$ and $\underset{\rightarrow}{\Omega}$ for those contexts that contain only left- and right-directed atoms, respectively. More precisely, these are generated by the grammars

$$\underset{\leftarrow}{\Omega}, \underset{\leftarrow}{\Delta} ::= \underset{\leftarrow}{\Omega}_1 \; \underset{\leftarrow}{\Omega}_2 \mid \cdot \mid \underset{\leftarrow}{a}^+ \qquad \text{and} \qquad \underset{\rightarrow}{\Omega}, \underset{\rightarrow}{\Delta} ::= \underset{\rightarrow}{\Omega}_1 \; \underset{\rightarrow}{\Omega}_2 \mid \cdot \mid \underset{\rightarrow}{a}^+ .$$

Positive propositions, A^+, are process expressions that reify run-time configurations Ω as static objects.

$$A^+, B^+ ::= \underset{\leftarrow}{a}^+ \mid \underset{\rightarrow}{a}^+ \mid A^+ \bullet B^+ \mid \mathbf{1} \mid \downarrow A^-$$

This reification is expressed as $\bullet\Omega = A^+$, as defined in the adjacent figure. This operation is the same as the one defined in fig. 5.9, except that atom directions are preserved.

$$\bullet \underset{\leftarrow}{a}^+ = \underset{\leftarrow}{a}^+$$
$$\bullet \underset{\rightarrow}{a}^+ = \underset{\rightarrow}{a}^+$$
$$\bullet(\Omega_1 \; \Omega_2) = (\bullet\Omega_1) \bullet (\bullet\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$\bullet A^- = \downarrow A^-$$

Figure 6.1: Reifying a configuration as a process

The propositions $\underset{\leftarrow}{a}^+$ and $\underset{\rightarrow}{a}^+$ are the expressions for left- and right-directed messages. The proposition $A^+ \bullet B^+$ reifies parallel, end-to-end composition of configurations: $A^+ \bullet B^+$ is now interpreted as the expression for a process that spawns a new process, A^+, and then continues as B^+. And $\mathbf{1}$ is interpreted as the expression for a process that immediately terminates, thereby reifying the empty configuration, $(\cdot)$. Lastly, the proposition $\downarrow A^-$ is interpreted as the expression for a quoted process: executing that expression will result in the running process A^-.

$\underset{\leftarrow}{a}^+$	*left-directed message*
$\underset{\rightarrow}{a}^+$	*right-directed message*
$A^+ \bullet B^+$	*parallel composition of* A^+ *and* B^+
$\mathbf{1}$	*terminating process*
$\downarrow A^-$	*quoted process*

Table 6.4: A formula-as-process interpretation of positive propositions

Notice that there is no propositional connective that corresponds to a *send* operation. Sending a message is instead accomplished by spawning a process that is itself only a message: $\underset{\leftarrow}{a}^+ \bullet B^+$, $B^+ \bullet \underset{\rightarrow}{a}^+$, $\underset{\rightarrow}{a}^+ \bullet B^+$, and $B^+ \bullet \underset{\leftarrow}{a}^+$ are all possible but built from other propositional forms. This is analogous to the way that the asynchronous π-calculus sends a message by parallel composition, as in $\bar{x}\langle y \rangle \mid P$. Treating send operations this way makes the formula-as-process ordered rewriting framework an asynchronous calculus.

6.1.1 *Focused ordered rewriting as message-passing communication*

The three tweaks introduced by the formula-as-process interpretation to the structure of propositions – especially atom directions and atomic premises for implications – trickle down through the right- and left-focus judgments used to define rewriting:

- First, because each positive atom is now marked with a direction, the $\textsc{id}^{a^+}$ rule that was previously part of the right-focus judgment's definition is replaced by two similar rules – one for each direction:

$$\frac{}{[\underset{\leftarrow}{a}^+] \dashv\Vdash \underset{\leftarrow}{a}^+}\ \textsc{id}^{\underset{\leftarrow}{a}^+} \qquad \text{and} \qquad \frac{}{[\underset{\rightarrow}{a}^+] \dashv\Vdash \underset{\rightarrow}{a}^+}\ \textsc{id}^{\underset{\rightarrow}{a}^+}.$$

  The other right-focusing rules remain unchanged.

$$\frac{}{[a^+] \dashv\Vdash a^+}\ \textsc{id}^{a^+}$$

- Second, because $\underset{\rightarrow}{a}^+ \setminus B^+$ and $B^- \,/\, \underset{\leftarrow}{a}^+$ are now the only valid forms of implications, the left-focus judgment and its rules may be refined. Instead of $\Omega_L\ [A^-]\ \Omega_R \Vdash C^+$, which has arbitrary contexts to the left and right of A^-, the judgment is now $\underset{\rightarrow}{\Omega}_L\ [A^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+$ – the left-hand context consists only of right-directed atoms, hence $\underset{\rightarrow}{\Omega}_L$; symmetrically, the right-hand context consists only of left-directed atoms, hence $\underset{\leftarrow}{\Omega}_R$. The left-focus rules for the left- and right-handed implications are also revised to

$$\frac{\underset{\rightarrow}{\Omega}_L\ [B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}^+\ [\underset{\rightarrow}{a}^+ \setminus B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \setminus\textsc{l}' \qquad \text{and} \qquad \frac{\underset{\rightarrow}{\Omega}_L\ [B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\ [B^- \,/\, \underset{\leftarrow}{a}^+]\ \underset{\leftarrow}{a}^+\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ /\textsc{l}',$$

  which are the derivable from the earlier $\setminus\textsc{l}$ and $/\textsc{l}$ rules, as shown in the adjacent figure.

$$\dfrac{\dfrac{}{[\underset{\rightarrow}{a}^+] \dashv\Vdash \underset{\rightarrow}{a}^+}\ \textsc{id}^{\underset{\rightarrow}{a}^+} \qquad \underset{\rightarrow}{\Omega}_L\ [B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}^+\ [\underset{\rightarrow}{a}^+ \setminus B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \setminus\textsc{l}$$
$$\wr$$
$$\frac{\underset{\rightarrow}{\Omega}_L\ [B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}^+\ [\underset{\rightarrow}{a}^+ \setminus B^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \setminus\textsc{l}'$$

Figure 6.2: Deriving the $\setminus\textsc{l}'$ left focus rule

  The other rules for the left-focus judgment remain fundamentally unchanged, save for the fact that the left- and right-hand contexts now contain only atoms of the complementary direction.

Figure 6.3: A formula-as-process ordered rewriting framework

$$\begin{array}{rl}
\textsc{positive props.} & A^+, B^+ ::= \underset{\leftarrow}{a}^+ \mid \underset{\rightarrow}{a}^+ \mid A^+ \bullet B^+ \mid \mathbf{1} \mid \downarrow A^- \\
\textsc{negative props.} & A^-, B^- ::= \underset{\rightarrow}{a}^+ \setminus B^- \mid B^- / \underset{\leftarrow}{a}^+ \mid A^- \mathbin{\&} B^- \mid \top \mid \uparrow A^+ \mid \hat{p}^- \\
\textsc{contexts} & \Omega, \Delta ::= \Omega_1\, \Omega_2 \mid \cdot \mid A^- \mid \underset{\leftarrow}{a}^+ \mid \underset{\rightarrow}{a}^+ \\
\textsc{signatures} & \Phi ::= \cdot \mid \Phi, \hat{p}^- \triangleq A^-
\end{array}$$

$$\frac{[A^+] \dashv\Vdash \Omega_1 \quad [B^+] \dashv\Vdash \Omega_2}{[A^+ \bullet B^+] \dashv\Vdash \Omega_1\, \Omega_2}\ \bullet\textsc{r} \qquad \frac{}{[\mathbf{1}] \dashv\Vdash \cdot}\ \mathbf{1}\textsc{r}$$

$$\frac{}{[\underset{\leftarrow}{a}^+] \dashv\Vdash \underset{\leftarrow}{a}^+}\ \textsc{id}^{\underset{\leftarrow}{a}^+} \qquad \frac{}{[\underset{\rightarrow}{a}^+] \dashv\Vdash \underset{\rightarrow}{a}^+}\ \textsc{id}^{\underset{\rightarrow}{a}^+} \qquad \frac{}{[\downarrow A^-] \dashv\Vdash A^-}\ \downarrow\textsc{r}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\, [B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, \underset{\rightarrow}{a}^+\, [\underset{\rightarrow}{a}^+ \setminus B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \setminus\textsc{l}' \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, [B^- / \underset{\leftarrow}{a}^+]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ /\textsc{l}'$$

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, [A^- \mathbin{\&} B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \&\textsc{l}_1 \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, [A^- \mathbin{\&} B^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}\ \&\textsc{l}_2 \qquad \text{(no } \top\textsc{l} \text{ rule)}$$

$$\frac{}{[\uparrow A^+] \Vdash A^+}\ \uparrow\textsc{l}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+ \quad [C^+] \dashv\Vdash \Omega'}{\underset{\rightarrow}{\Omega}_L\, A^-\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ \longrightarrow\textsc{i} \qquad \frac{\Omega \longrightarrow \Omega'}{\Omega_L\, \Omega\, \Omega_R \longrightarrow \Omega_L\, \Omega'\, \Omega_R}\ \longrightarrow\textsc{c}$$

$$\frac{}{\Omega \Longrightarrow \Omega}\ \Longrightarrow\textsc{r} \qquad \frac{\Omega \longrightarrow \Omega' \quad \Omega' \Longrightarrow \Omega''}{\Omega \Longrightarrow \Omega''}\ \Longrightarrow\textsc{t}$$

Having refined the left-focus judgment to use message contexts, we may similarly refine the principal reduction rule, $\longrightarrow\textsc{i}$:

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash B^+ \quad [B^+] \dashv\Vdash \Omega'}{\underset{\rightarrow}{\Omega}_L\, A^-\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ \longrightarrow\textsc{i}.$$

The compatibility rule, $\longrightarrow\textsc{c}$, remains unchanged. Figure 6.3 summarizes the revised rules for the formula-as-process ordered rewriting framework.

6.1.2 *Comments*

Now we are in a position to understand how the two principal syntactic changes – atom directions and atomic premises for implications – combine to endow the otherwise untyped processes with a modicum of static typing.

In the expression $\downarrow A^- \bullet \underset{\rightarrow}{a}^+$, the atom $\underset{\rightarrow}{a}^+$ is an outgoing message, owing to its direction away from the (quoted) process A^-. If the premises of left- and right-handed implications were *not* restricted to atoms of complementary direction,

then A^- might possibly be the input process $\uparrow\downarrow B^- \,/\, \underrightarrow{a}^+$, which could incorrectly (re-)capture the outgoing message, $\underrightarrow{a}^+$, that it just sent:

$$(\uparrow\downarrow B^- \,/\, \underrightarrow{a}^+) \bullet \underrightarrow{a}^+ \longrightarrow (\uparrow\downarrow B^- \,/\, \underrightarrow{a}^+)\ \underrightarrow{a}^+ \longrightarrow B^- \,.$$

However, because the premises of left- and right-handed implications are indeed restricted to atoms of complementary direction, this scenario is impossible – $\uparrow\downarrow B^- \,/\, \underrightarrow{a}^+$ is not even a well-formed proposition!

Formally, this property that outgoing messages cannot be recaptured is established the following theorem.

THEOREM 6.1. *If* $\Omega = \underleftarrow{\Omega}_L\ \Omega_0\ \underrightarrow{\Omega}_R \Longrightarrow \Omega'$, *then* $\Omega' = \underleftarrow{\Omega}_L\ \Omega_0'\ \underrightarrow{\Omega}_R$ *for some context* Ω_0' *such that* $\Omega_0 \Longrightarrow \Omega_0'$.

Proof. By induction on the structure of the given rewriting sequence, using inversion on the individual rewriting steps. □

As a related consequence of these syntactic restrictions, there is no contention for messages. Without these restrictions, the above trace could be adapted to one in which a race could arise between two processes contending for the same message:

$$(\uparrow\downarrow B^- \,/\, \underrightarrow{a}^+)\ \underrightarrow{a}^+\ (\underrightarrow{a}^+ \setminus \uparrow\downarrow C^-) \begin{array}{l} \nearrow\ B^- \\ \searrow\ C^- \end{array}$$

However, with these restrictions in place, there is no such message – neither $\underleftarrow{a}^+$ nor $\underrightarrow{a}^+$ – that can cause contention between $\uparrow B^- \,/\, \underleftarrow{a}^+$ and $\underrightarrow{a}^+ \setminus \uparrow C^-$ because $\underleftarrow{a}^+ \neq \underrightarrow{a}^+$, that is, there is no one atom that may have both directions.

Even with the restriction of left- and right-handed implication premises to atoms of complementary direction, it is nevertheless possible for a process to send *itself* a message, as in

$$(\uparrow\downarrow B^- \,/\, \underleftarrow{a}^+) \bullet \underleftarrow{a}^+ \longrightarrow (\uparrow\downarrow B^- \,/\, \underleftarrow{a}^+)\ \underleftarrow{a}^+ \longrightarrow B^- \,.$$

But this is not troubling because the intended recipient – the process itself – does indeed receive the message.

ORDERED CONJUNCTIONS are dual to the left- and right-handed implications. So one might think that ordered conjunctions ought to be restricted to those of the form $\underleftarrow{a}^+ \bullet B^+$ and $B^+ \bullet \underrightarrow{a}^+$, as a kind of dual restriction to those placed on implications. Just as the implications are restricted to receive only incoming messages, these ordered conjunctions would restrict processes to sending only outgoing messages.

Although certainly possible, such restrictions would limit the expressiveness of formula-as-process ordered rewriting by precluding a process from sending itself a message – $(\uparrow\downarrow B^- \,/\, \underleftarrow{a}^+) \bullet \underleftarrow{a}^+$ would not be well-formed, for example. Moreover, in chapter 10, we will present a correspondence between

ordered propositions and the not-yet-introduced singleton proofs (chapters 8 and 9), which will turn out to be more direct if we retain the general $A^+ \bullet B^+$ form for ordered conjunctions. And finally, the general $A^+ \bullet B^+$ form is more in line with the asynchronous nature of ordered rewriting. For all of these reasons, we choose not to impose any restrictions on ordered conjunctions.

6.1.3 *Coinductively defined negative propositions*

Recall from chapter 5 that ordered rewriting is terminating: for all ordered contexts Ω, every rewriting sequence from Ω is finite (theorem 5.8). Although a seemingly pleasant property, termination significantly limits the expressiveness of focused ordered rewriting. For example, without unbounded rewriting, we cannot even describe producer–consumer systems or finite automata.

As the proof of termination shows, rewriting is bounded precisely because contexts consist of finitely many *finite* propositions. In multiset and ordered rewriting, unbounded behavior is traditionally introduced by way of persistent propositions that may be replicated as much as needed.[5] This is related to Milner's use of replication, $!P$, in the π-calculus.[6] (See section 3.3 for a more detailed discussion of replication.)

[5] Polakow 2001; Watkins et al. 2002; Simmons 2012.

[6] Milner 1999.

However, another option – and the one that we pursue here – is to permit circular negative propositions in the form of mutually coinductive definitions, $\hat{p}^- \triangleq A^-$, where the grammar of negative propositions is extended to include these coinductively defined propositions:

$$A^-, B^- ::= \underset{\rightarrow}{a}^+ \setminus B^- \mid B^- / \underset{\leftarrow}{a}^+ \mid A^- \mathbin{\&} B^- \mid \top \mid \hat{p}^- \,.$$

Sequent calculi with recursive definitions of this kind have been studied previously,[7] but, to the best of our knowledge, the use of coinductive definitions in the context of logically motivated rewriting systems is new.

[7] Hallnäs 1991; Eriksson 1991; Schroeder-Heister 1993; McDowell and Miller 2000; Tiu and Momigliano 2012.

That the definitions $\hat{p}^- \triangleq A^-$ are indeed coinductive is guaranteed by imposing the requirement that along every cycle among defined propositions there is a logical connective.[8] For example, the definition $\hat{p}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \hat{p}^-$ or even the definitions $\hat{p}^- \triangleq \hat{q}^-$ and $\hat{q}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \hat{p}^-$ are acceptable because $(\underset{\rightarrow}{a}^+ \setminus -)$ occurs along the cycle from $\hat{p}^-$ to itself; but the definitions $\hat{p}^- \triangleq \hat{q}^-$ and $\hat{q}^- \triangleq \hat{p}^-$ are forbidden because no logical connective occurs along the cycle.

[8] This generalizes the local *contractivity* condition described by Gay and Hole (2005).

To clarify, these definitions are coinductive in only a syntactic sense; when interpreted as processes, they are not (necessarily) behaviorally coinductive. For example, the proposition $\hat{p}^-$ given by $\hat{p}^- \triangleq \underset{\rightarrow}{a} \setminus \underset{\rightarrow}{a} \bullet \hat{p}^-$ does not correspond to a productive process when viewed from the formula-as-process perspective – after receiving an initial message $\underset{\rightarrow}{a}$, the process $\hat{p}^-$ diverges without sending or receiving any further messages. Therefore, our coinductively defined propositions are *not* greatest fixed points (or coinductive proofs) in the sense of Fortier and Santocanale.[9]

[9] Fortier and Santocanale 2013.

Coinductive definitions are collected into a signature, Φ, that indexes the rewriting relations: $\longrightarrow_\Phi$ and $\Longrightarrow_\Phi$.[10] Syntactically, these signatures are given

[10] We often elide the index, as it is usually clear from context.

by the grammar

$$\Phi ::= \cdot \mid \Phi, (\hat{p}^- \triangleq A^-)\,.$$

A signature Φ is well-formed if every $\hat{p}^-$ that occurs in the body of a definition itself has a definition in Φ.

By analogy with recursive types from functional programming[11], we must then decide whether to treat the coinductive definitions $\hat{p}^- \triangleq A^-$ *iso*recursively or *equi*recursively. Under an equirecursive treatment, definitions may be silently unrolled or rolled at will; in other words, $\hat{p}^-$ is literally *equal* to its unrolling: $\hat{p}^- = A^-$. In contrast, under an isorecursive treatment, unrolling a coinductively defined proposition would count only as an explicit rule for the left-focus judgment: $\hat{p}^- \neq A^-$ but the adjacent $\triangleq\text{L}$ rule would be present.

Because these coinductively defined propositions are not generative, there is not much difference between their equirecursive and isorecursive treatments.[12] We choose an equirecursive treatment of definitions simply because the accompanying generous notion of equality helps to minimize the conceptual overhead of coinductively defined propositions.

[11] See Pierce (2002), for example.

$$\frac{((\hat{p}^- \triangleq A^-) \in \Phi) \quad \underset{\rightarrow}{\Omega}_L\,[A^-]\,\underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+}{\underset{\rightarrow}{\Omega}_L\,[\hat{p}^-]\,\underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+}\ \triangleq\text{L}$$

[12] Amadio and Cardelli 1993.

How do these coinductively defined negative propositions interact with the left-focus judgment, which is defined inductively? The answer is that not all coinductively defined propositions can be successfully put into focus. As previously mentioned, the proposition $\hat{p}^-$ given by $\hat{p}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \hat{p}^-$ is certainly a well-defined coinductive proposition, owing to the existence of $(\underset{\rightarrow}{a}^+ \setminus -)$ along the cycle. Yet it cannot be successfully put into left focus – there are no contexts $\underset{\rightarrow}{\Omega}_L$ and $\underset{\leftarrow}{\Omega}_R$ and positive consequent C^+ for which $\underset{\rightarrow}{\Omega}_L\,[\hat{p}^-]\,\underset{\leftarrow}{\Omega}_R \Vdash C^+$ is derivable. To derive a left-focus judgment on $\hat{p}^-$, the finite context $\underset{\rightarrow}{\Omega}_L$ would need to hold an infinite stream of $\underset{\rightarrow}{a}^+$ atoms – an impossible feat for the inductively defined, and hence finite, contexts like $\underset{\rightarrow}{\Omega}_L$ that we consider here.[13]

However, by inserting $\uparrow\downarrow$ as a double shift to blur focus – in a way similar to how double shifts were used in the embedding of unfocused rewriting (section 5.3.1) – the definition can be revised to one that admits a left-focus judgment. Specifically, if $\hat{p}^-$ is instead given by $\hat{p}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \uparrow\downarrow\hat{p}^-$, then $\underset{\rightarrow}{a}^+\,[\hat{p}^-] \Vdash \downarrow\hat{p}^-$ is derivable, and so $\underset{\rightarrow}{a}^+\,\hat{p}^- \longrightarrow \hat{p}^-$. More generally, any coinductively defined proposition that has an $\uparrow$ shift along *some* cycle can be successfully put into focus.

One might consider elevating the $\uparrow$-shift property to a requirement on coinductively defined propositions – *i.e.*, demanding that every coinductively proposition have an $\uparrow$ shift along some cycle. This would forbid any definitions that cannot be put into left focus, such as $\hat{p}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \hat{p}^-$. Although perhaps well-intentioned, such a requirement seems somewhat under-motivated after observing that even the proposition $\top$ cannot be successfully put into focus.

[13] Baelde and Miller (2007) present a focusing system for least and greatest fixed points in linear logic that allows focus on fixed points such as $\hat{p}^- \triangleq \underset{\rightarrow}{a}^+ \setminus \hat{p}^-$ to either continue or blur. We do not pursue that generalization, nor a generalization to circular focusing phases (which would necessitate infinite contexts).

6.2 *A local interaction semantics*

For the formula-as-process interpretation, we have thus far examined the rewriting judgment, $\Omega \longrightarrow \Omega'$, and suggested that it represents a kind of reduction semantics for the underlying processes.

But a reduction semantics is not the only way to describe the operational semantics of a process calculus. For example, in the π-calculus, labeled transition systems are frequently used as an alternative to a reduction semantics, particularly when an understanding of how processes interact with their surroundings is needed.

For the formula-as-process ordered rewriting framework, we can similarly conceive of a local interaction semantics of this sort. All communication occurs through message passing, so there are just two ways a process configuration can interact with its surrounding environment – either send messages or receive them; either make an output transition or make an input transition. A process configuration can also forgo interacting with its environment and make a silent, internal transition as the configuration's components interact with each other.

Traditionally, these three forms of transition – internal, output, and input – are expressed with a unified labeled transition judgment in which the labels distinguish among the three forms of transition. Here we instead prefer to use distinct syntax for each form of transition.

INTERNAL TRANSITIONS In labeled transition systems for process calculi, internal τ-transitions express interaction of a process configuration, not with its environment, but within itself among its constituent processes. In the π-calculus, these internal transitions coincide with the notion of reduction but are defined as $\overset{\tau}{\longrightarrow}$ in such a way that the explicit and sometimes cumbersome structural congruence is not needed, thereby simplifying proofs.

In our setting, however, we have no explicit structural congruence; the implicit monoid laws do not complicate proofs, so we can get away without defining a distinct notion of internal τ-transition. Whenever we want to describe an internal transition, the $\longrightarrow$ reduction relation can be used instead.

In the π-calculus, there is also a notion of weak internal τ-transition, $\overset{\tau}{\Longrightarrow}$, which is the reflexive, transitive closure of $\overset{\tau}{\longrightarrow}$. Just as we use the $\longrightarrow$ reduction relation whenever we want to describe an internal transition, we will use its reflexive, transitive closure, $\Longrightarrow$, whenever we want to describe a weak internal transition.

OUTPUT TRANSITIONS Similar to our treatment of internal transitions, we do not adopt an explicit judgment for output interactions but instead make use of context equality. We say that the context Ω outputs messages $\underset{\leftarrow}{\Omega}_L$ to the left and messages $\underset{\rightarrow}{\Omega}_R$ to the right exactly when $\Omega = \underset{\leftarrow}{\Omega}_L\ \Omega'\ \underset{\rightarrow}{\Omega}_R$ for some context Ω'. This equality expresses an immediate, or strong, output of messages $\underset{\leftarrow}{\Omega}'_L$

and Ω'_R from the context Ω.[14] We will sometimes refer to the context Ω' here as the *continuation context* because it represents the context that remains after the output of Ω_L and Ω_R occurs.

As an example, both $a\, b\, C^-$ and $a\, C^-\, b$ output a to the left (and nothing to the right), but more precisely, the former outputs $a\, b$, whereas the latter does not output b at all.

In addition to immediate, or strong, output transitions, it is also typical in a labeled transition semantics to express eventual, or weak, output transitions. A weak output transition consists of finitely many internal transitions, followed by a single strong output transition, along with finitely many internal transitions on the continuation.

In the π-calculus, the weak output transition relation is $\stackrel{\bar{x}\langle y\rangle}{\Longrightarrow} = \Longrightarrow\stackrel{\tau}{\longrightarrow}\Longrightarrow\stackrel{\bar{x}\langle y\rangle}{\longrightarrow}\Longrightarrow\stackrel{\tau}{\longrightarrow}\Longrightarrow$. In the formula-as-process ordered rewriting framework, a weak output transition could be expressed by $\Omega \Longrightarrow \Omega'_L\ \Omega'_0\ \Omega'_R$ and $\Omega'_0 \Longrightarrow \Omega'$ together – the context Ω eventually outputs Ω'_L and Ω'_R and eventually arrives at the continuation context Ω'. Based on theorem 6.1, we can more concisely express the same weak output transition as $\Omega \Longrightarrow \Omega'_L\ \Omega'\ \Omega'_R$. This is the form in which we will usually express weak output transitions.

[14] It is roughly analogous to $\stackrel{x\langle y\rangle}{\longrightarrow}$, the π-calculus's output transition relation.

INPUT TRANSITIONS Unlike internal and output transitions, we use an explicit judgment for input interactions. The judgment $\Omega_L\ [\Omega]\ \Omega_R \longrightarrow \Omega'$ indicates that, upon receiving messages Ω_L from the left and Ω_R from the right, the context Ω may evolve to Ω' in a single step.[15] In other words, for each such judgment there should be a corresponding reduction:

[15] It is roughly analogous to $\stackrel{x(y)}{\longrightarrow}$, the π-calculus's input transition relation.

THEOREM 6.2 (Soundness). *If* $\Omega_L\ [\Omega]\ \Omega_R \longrightarrow \Omega'$, *then* $\Omega_L\ \Omega\ \Omega_R \longrightarrow \Omega'$.

In terms of the judgment's input/output mode, Ω is the sole input to the judgment, whereas it produces the contexts Ω_L, Ω_R, and Ω' as outputs of the judgments. Thus, the input transition judgment answers the question "What input messages suffice for Ω to make a reduction?"

AS THE NOTATION is intended to suggest, each input transition at its heart derives from focusing on a single negative proposition, A^-, as captured by the $[]{\longrightarrow}\text{I}$ rule:[16]

$$\frac{\Omega_L\ [A^-]\ \Omega_R \Vdash C^+ \qquad [C^+] \dashv\Vdash \Omega'}{\Omega_L\ [A^-]\ \Omega_R \longrightarrow \Omega'}\ []{\longrightarrow}\text{I}\,.$$

[16] Notice that it is quite possible in this rule for both Ω_L and Ω_R to be empty and for the judgment to express the input of no messages at all. But that happens only if A^- has an $\uparrow$ shift as its top-level connective.

Aside from the change of judgment in the rule's conclusion, this $[]{\longrightarrow}\text{I}$ rule is identical to the core ${\longrightarrow}\text{I}$ rule for reduction. How can we claim that the input transition judgment is distinct from the reduction judgment?

The difference between the judgments is twofold. First, and most importantly, this input transition differs from a reduction in terms of its input/output mode. In a reduction $\Omega_L\ A^-\ \Omega_R \longrightarrow \Omega'$, the entire $\Omega_L\ A^-\ \Omega_R$ context is treated as an input to the reduction judgment, and Ω' is treated as an output made by the judgment. In the input transition $\Omega_L\ [A^-]\ \Omega_R \longrightarrow \Omega'$, on the other hand,

$$\frac{\underset{\rightarrow}{\Omega}_L\ [A^-]\ \underset{\leftarrow}{\Omega}_R \Vdash C^+ \quad [C^+] \dashv\Vdash \Omega'}{\underset{\rightarrow}{\Omega}_L\ [A^-]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ []{\longrightarrow}\text{C}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}\ [\Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega' \quad (\underset{\rightarrow}{\Omega}_L \neq (\cdot) \text{ or } \underset{\leftarrow}{\Omega}_R \neq (\cdot))}{\underset{\rightarrow}{\Omega}_L\ [\underset{\rightarrow}{a}\ \Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ [\underset{\rightarrow}{a}]\text{C}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\ [\Omega]\ \underset{\leftarrow}{a}\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega' \quad (\underset{\rightarrow}{\Omega}_L \neq (\cdot) \text{ or } \underset{\leftarrow}{\Omega}_R \neq (\cdot))}{\underset{\rightarrow}{\Omega}_L\ [\Omega\ \underset{\leftarrow}{a}]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ [\underset{\leftarrow}{a}]\text{C}$$

$$\frac{[\Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{[\omega\ \Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \omega\ \Omega'}\ [\omega]\text{C}_1 \qquad \frac{\underset{\rightarrow}{\Omega}_L\ [\Omega] \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\ [\Omega\ \omega] \longrightarrow \Omega'\ \omega}\ [\omega]\text{C}_2$$

Figure 6.4: An input transition judgment

only the proposition A^- is treated as an input to the judgment, and the contexts $\underset{\rightarrow}{\Omega}_L$, $\underset{\leftarrow}{\Omega}_R$, and Ω' are all treated as outputs made by the input transition judgment.

The second difference is that, unlike the reduction judgment, the input transition judgment is enriched with several other rules. In addition to the core input transition rule, $[]{\longrightarrow}\text{I}$, other compatibility rules exist.

Two of these rules allow the external inputs expected by an input transition to be (partially) satisfied internally by the context itself.

$$\frac{\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}\ [\Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega' \quad (\underset{\rightarrow}{\Omega}_L \neq (\cdot) \text{ or } \underset{\leftarrow}{\Omega}_R \neq (\cdot))}{\underset{\rightarrow}{\Omega}_L\ [\underset{\rightarrow}{a}\ \Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ [\underset{\rightarrow}{a}]\text{C}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\ [\Omega]\ \underset{\leftarrow}{a}\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega' \quad (\underset{\rightarrow}{\Omega}_L \neq (\cdot) \text{ or } \underset{\leftarrow}{\Omega}_R \neq (\cdot))}{\underset{\rightarrow}{\Omega}_L\ [\Omega\ \underset{\leftarrow}{a}]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}\ [\underset{\leftarrow}{a}]\text{C}$$

For example, the $[\underset{\rightarrow}{a}]\text{C}$ rule: if Ω can reduce to Ω' upon input of surrounding $\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}$ and $\underset{\leftarrow}{\Omega}_R$, then $\underset{\rightarrow}{a}\ \Omega$ can reduce to Ω' upon input of surrounding $\underset{\rightarrow}{\Omega}_L$ and $\underset{\leftarrow}{\Omega}_R$. In other words, in the context $\underset{\rightarrow}{a}\ \Omega$, the atom $\underset{\rightarrow}{a}$ already, internally satisfies Ω's demand for $\underset{\rightarrow}{a}$.[17] The $[\underset{\leftarrow}{a}]\text{C}$ rule is symmetric, involving $\underset{\leftarrow}{a}$ on the right. Algebraically, these two rules express a form of associativity.

Read top-down, these $[\underset{\rightarrow}{a}]\text{C}$ and $[\underset{\leftarrow}{a}]\text{C}$ rules allow an input message to be absorbed by an input transition. In addition, the input transition judgment is equipped with several (limited) compatibility rules. Instead of absorbing a message like the $[\underset{\rightarrow}{a}]\text{C}$ and $[\underset{\leftarrow}{a}]\text{C}$ rules do, these compatibility rules frame a message or process ω onto an input transition, passing ω through.[18]

$$\frac{[\Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{[\omega\ \Omega]\ \underset{\leftarrow}{\Omega}_R \longrightarrow \omega\ \Omega'}\ [\omega]\text{C}_1 \qquad \frac{\underset{\rightarrow}{\Omega}_L\ [\Omega] \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\ [\Omega\ \omega] \longrightarrow \Omega'\ \omega}\ [\omega]\text{C}_2$$

Notice that these rules apply only to one-sided input transitions: Ω must require no inputs at the side at which ω is added. This is because these rules pass ω through the input transition unaffected, and so ω serves as an interaction barrier at the end at which it appears.

The full complement of input transition rules is summarized in fig. 6.4.

[17] The side conditions in these rules serve to ensure that the last input atom cannot be absorbed into the context. This way, each input transition that derives from an implication cannot degenerate to a reduction. (Input tensions that derive from propositions $\uparrow B^+$ will require no inputs, however.)

[18] Recall from section 6.1 that $\omega ::= \underset{\rightarrow}{a} \mid \underset{\leftarrow}{a} \mid A^-$.

Now we may finally prove the previously stated claim of soundness for input transitions – that each input transition has a corresponding reduction.

THEOREM 6.3 (Soundness). *If* $\underrightarrow{\Omega}_L\,[\Omega]\,\underleftarrow{\Omega}_R \longrightarrow \Omega'$, *then* $\underrightarrow{\Omega}_L\,\Omega\,\underleftarrow{\Omega}_R \longrightarrow \Omega'$.

Proof. By induction on the structure of the given input transition. □

Also, output and input transitions are together complete, in the sense that each reduction can be broken down into an input transition with complementary output transitions:

THEOREM 6.4 (Completeness). *If* $\Omega \longrightarrow \Omega'$, *then there exist contexts* Ω'_L, $\underrightarrow{\Omega}_L$, Ω_0, $\underleftarrow{\Omega}_R$, Ω'_R, *and* Ω'_0 *such that:* $\Omega = (\Omega'_L\,\underrightarrow{\Omega}_L)\,\Omega_0\,(\underleftarrow{\Omega}_R\,\Omega'_R)$ *and* $\underrightarrow{\Omega}_L\,[\Omega_0]\,\underleftarrow{\Omega}_R \longrightarrow \Omega'_0$ *and* $\Omega' = \Omega'_L\,\Omega'_0\,\Omega'_R$.

Proof. By induction on the structure of the given reduction. □

Together, these soundness and completeness results may also be thought of as establishing the admissibility and invertibility of the following rule.

$$\frac{\Omega = \Omega_L\,\Delta\,\Omega_R \quad \Omega_L = \Omega'_L\,\underrightarrow{\Delta}_L \quad \underrightarrow{\Delta}_L\,[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta' \quad \underleftarrow{\Delta}_R\,\Omega'_R = \Omega_R \quad \Omega'_L\,\Delta'\,\Omega'_R = \Omega'}{\Omega \longrightarrow \Omega'}$$

The following lemma will also prove useful later.

LEMMA 6.5. *If* $\underrightarrow{\Delta}_L\,[\underrightarrow{a}\,\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$, *then either:* $\underrightarrow{\Delta}_L\,\underrightarrow{a}\,[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$; *or* $\underrightarrow{\Delta}_L$ *is empty and* $\Delta' = \underrightarrow{a}\,\Delta'_0$ *and* $[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'_0$, *for some* Δ'_0. *Symmetrically, if* $\underrightarrow{\Delta}_L\,[\Delta\,\underleftarrow{a}]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$, *then either:* $\underrightarrow{\Delta}_L\,[\Delta]\,\underleftarrow{a}\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$; *or* $\underleftarrow{\Delta}_R$ *is empty,* $\Delta' = \Delta'_0\,\underleftarrow{a}$, *and* $\underrightarrow{\Delta}_L\,[\Delta] \longrightarrow \Delta'_0$, *for some* Δ'_0.

Proof. By induction on the structure of the given derivation. □

We could also consider a notion of eventual, or weak, input transition, akin to the π-calculus's $\overset{x(y)}{\Longrightarrow}$ relation. There is not an especially concise way to express weak input transitions using our notation for strong input transitions, and weak input transitions in and of themselves will not prove to be particularly useful to us, so we do not pursue them here.

6.3 *Choreographing string rewriting specifications*

So far in this chapter, we have presented a formula-as-process refinement of the focused ordered rewriting framework and given it a local interaction semantics based on an implicit labeled transition system for output and input transitions. With this local interaction semantics in hand, we can return to our goal of assigning a concrete operational semantics to string rewriting specifications. We will show how to operationalize, or *choreograph*, these specifications by embedding them within the formula-as-process ordered rewriting framework.

To choreograph a string rewriting specification (Σ, Θ), we would like to map each symbol $a \in \Sigma$ to a proposition such that the string rewriting axioms Θ are mapped to derivable rewritings in our formula-as-process ordered rewriting framework. In other words, to choreograph (Σ, Θ), we would like to find a map θ from symbols to propositions and a signature Φ of coinductive definitions such that θ is a witness to the (strong) bisimilarity of string rewriting under the axioms Θ and formula-as-process ordered rewriting under the definitions Φ. That is, we would like to find a pair (θ, Φ) for which we can complete the diagrams

$$\begin{array}{ccc} w & \longrightarrow_{\Theta} & w' \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \Omega & \dashrightarrow_{\Phi} & \Omega' \end{array} \qquad \text{and} \qquad \begin{array}{ccc} w & \dashrightarrow_{\Theta} & w' \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \Omega & \longrightarrow_{\Phi} & \Omega' \end{array},$$

where $w \xrightarrow{\theta} \Omega$ holds exactly when $\Omega = \theta(w)$. Only if the pair (θ, Φ) satisfies these diagrams does it constitute a *choreography* of the specification (Σ, Θ).[19]

[19] An arbitrary pair (θ, Φ) might be called a *pre-choreography*.

Because ordered rewriting in our formula-as-process framework permits only sensibly local interactions, we can be sure that the choreography (θ, Φ) explains the *how*, not just the what, of the concurrent system's dynamics. The map θ is key to the how. It serves as a *role assignment* for the string rewriting symbols, casting each symbol $a \in \Sigma$ in the role of either a message, $\underset{\leftarrow}{a}$ or $\underset{\rightarrow}{a}$, or a coinductively defined process, $\hat{a}$. (More formally, we will require that role assignments be injective monoid homomorphisms with this property.)

For a given specification, there will often be several role assignments that give rise to distinct choreographies, each one implying a different message-passing operationalization of the specification. Without applying other, external criteria, no one choreography has more desirable lower-level behavior than another – only the programmer is in a position to choose among choreographies.

Most of the $3^{|\Sigma|}$ role assignments for a specification's alphabet do not lead to adequate choreographies. Sometimes none of the possible role assignments produce a choreography.

6.3.1 *Choreographies by example*

Recall from chapter 4 the string rewriting specification (Σ, Θ) of a system that can rewrite strings over $\Sigma = \{a, b\}$ into the empty string if the initial string ends in b.

$$\Sigma = \{a, b\}$$
$$\Theta = (a \longrightarrow b), (b \longrightarrow \epsilon)$$

Let θ be the injective monoid homomorphism generated by mapping a to the right-directed message $\underset{\rightarrow}{a}$ and b to the coinductively defined process $\hat{b}$. The map θ is indeed a role assignment, but does it yield a meaningful choreography for the specification (Σ, Θ)?

$\theta = \{a \mapsto \underset{\rightarrow}{a}, b \mapsto \hat{b}\}$

We must determine if $\hat{b}$ can be given a definition $\Phi = (\hat{b} \triangleq B^-)$ such that the above, strong bisimulation diagrams can be completed. Because θ is injective, those diagrams are equivalent to the following ones: In the first diagram, the right-hand edge $w' \overset{\theta}{\text{---}} \Omega'$ can be replaced with $w' \overset{\theta}{\text{---}} \theta(w')$, but we cannot make a similar replacement for the second diagram because θ is not bijective, only injective.

$$\begin{array}{ccc} w & \longrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \theta(w) & \dashrightarrow_\Phi & \theta(w') \end{array} \qquad \text{and} \qquad \begin{array}{ccc} w & \dashrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \vdots{\scriptstyle\theta} \\ \theta(w) & \longrightarrow_\Phi & \Omega' \,. \end{array}$$

The first diagram gives us a way forward to a choreography: for each axiom $(w \longrightarrow w') \in \Theta$, the rewriting $\theta(w) \longrightarrow_\Phi \theta(w')$ must be derivable under the definitions Φ. In other words, these rewritings serve as constraints upon the definitions Φ that must be fulfilled if (θ, Φ) is to be a meaningful choreography for the specification (Σ, Θ).

In this example, the axioms $a\,b \longrightarrow b$ and $b \longrightarrow \epsilon$ induce the constraints

$$\underset{\rightarrow}{a}\,\hat{b} \dashrightarrow_\Phi \hat{b} \qquad \text{and} \qquad \hat{b} \dashrightarrow_\Phi (\cdot)\,.$$

Well, a definition $\hat{b} \triangleq \underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}$ would satisfy the first constraint but not the second, because $\underset{\rightarrow}{a}\,\hat{b} = \underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \longrightarrow_\Phi \hat{b}$. And a definition $\hat{b} \triangleq \uparrow\mathbf{1}$ would satisfy the second constraint but not the first, because $\hat{b} = \uparrow\mathbf{1} \longrightarrow_\Phi (\cdot)$. Fortunately, we can form a kind of greatest lower bound of these definitions using alternative conjunction[20]: the definition $\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \,\&\, \uparrow\mathbf{1}$ satisfies *both* constraints,

$$\underset{\rightarrow}{a}\,\hat{b} = \underset{\rightarrow}{a}\,((\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \,\&\, \uparrow\mathbf{1}) \longrightarrow_\Phi \hat{b} \qquad \text{and} \qquad \hat{b} = (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \,\&\, \uparrow\mathbf{1} \longrightarrow_\Phi (\cdot)\,.$$

And the second diagram holds because of the universal properties of the logical connectives.

$$\begin{array}{ccc} a\,b & \longrightarrow_\Theta & b \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \theta(a\,b) & = \underset{\rightarrow}{a}\,\hat{b} \dashrightarrow_\Phi \hat{b} = & \theta(b) \end{array}$$

$$\begin{array}{ccc} b & \longrightarrow_\Theta & \epsilon \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \theta(b) & = \hat{b} \dashrightarrow_\Phi (\cdot) = & \theta(\epsilon) \end{array}$$

Figure 6.5: Axioms induce rewritings as constraints on a choreography

[20] This is possible because the left-focus rule for alternative conjunction preserves focus.

NOT ALL ROLE ASSIGNMENTS yield meaningful choreographies, however. This happens when there is no solution to the constraints on Φ induced by the axioms and chosen role assignment. For a set of constraints to be satisfiable, three conditions must hold.

- *Each induced rewriting must have at least one process in its premise.* In the above example, for instance, role assignments θ' such that either $b \mapsto \underset{\leftarrow}{b}$ or $b \mapsto \underset{\rightarrow}{b}$ do not yield meaningful choreographies. Under such assignments, the axiom $b \longrightarrow_\Theta \epsilon$ induces either $\underset{\leftarrow}{b} \dashrightarrow_{\Phi'} (\cdot)$ or $\underset{\rightarrow}{b} \dashrightarrow_{\Phi'} (\cdot)$ as constraints. There are, however, no definitions that satisfy either constraint because the formula-as-process framework has no rules that permit an atom alone to be rewritten: messages are passive objects.
- *Each induced rewriting must have at most one process in its premise.* In the above example, for instance, the role assignment θ' such that $a \mapsto \hat{a}$ and

$b \mapsto \hat{b}$ does not yield a meaningful choreography. The axiom $a\,b \longrightarrow b$ induces the constraint $\hat{a}\,\hat{b} \dashrightarrow_{\Phi'} \hat{b}$. There are, however, no definitions for $\hat{a}$ and $\hat{b}$ that satisfy this constraint because the formula-as-process framework proscribes implications from having non-atomic premises: a process can input only messages, not other processes.

- *Each message in a premise must be directed inward, toward the premise's process.* In the above example, for instance, the role assignment θ' such that $a \mapsto \underset{\leftarrow}{a}$ and $b \mapsto \hat{b}$ does not yield a meaningful choreography. The axiom $a\,b \longrightarrow b$ induces the constraint $\underset{\leftarrow}{a}\,\hat{b} \dashrightarrow_{\Phi'} \hat{b}$. There is, however, no definition for $\hat{b}$ that satisfies this constraint because the formula-as-process framework requires that implications have atomic premises of *complementary* direction: a process can only receive messages intended for itself.

More generally, these observations suggest that only constraints of the form $\underset{\rightarrow}{\Omega}_L\,\hat{a}\,\underset{\leftarrow}{\Omega}_R \dashrightarrow_{\Phi} \Omega'$ are satisfiable, and that these constraints are induced by axioms of the form $w_1\,a\,w_2 \longrightarrow w'$. In the following section, we leverage these ideas to present a more formal description of the above procedure for choreographing string rewriting specifications within the formula-as-process ordered rewriting framework.

6.3.2 *A formal description of choreographing specifications*

To give a formal description of choreographing specifications, we define a judgment $\theta \vdash \Theta \rightsquigarrow \Phi$ that, when given a string rewriting specification (Σ, Θ) and a role assignment θ, yields formula-as-process definitions Φ that make string rewriting under Θ and formula-as-process ordered rewriting under Φ bisimilar, if such definitions exist:

$$\theta \vdash \Theta \rightsquigarrow \Phi \quad \text{only if} \quad \begin{array}{ccc} w & \longrightarrow_{\Theta} & w' \\ {\scriptstyle\theta}\big\downarrow & & \big\downarrow{\scriptstyle\theta} \\ \theta(w) & \dashrightarrow_{\Phi} & \theta(w') \end{array} \quad \text{and} \quad \begin{array}{ccc} w & \dashrightarrow_{\Theta} & w' \\ {\scriptstyle\theta}\big\downarrow & & \vdots{\scriptstyle\theta} \\ \theta(w) & \longrightarrow_{\Phi} & \Omega' \,. \end{array}$$

Stated differently, the judgment will be such that the following adequacy result will hold: "If $\theta \vdash \Theta \rightsquigarrow \Phi$, then $\theta(w) \longrightarrow_{\Phi} \Omega'$ if, and only if, there exists a string w' such that $w \longrightarrow_{\Theta} w'$ and $\theta(w') = \Omega'$."

This principal judgment also relies on an auxiliary elaboration judgment, $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ \,/\, \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-$, which we describe first.

The auxiliary judgment $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ \,/\, \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-$ elaborates the quasi-proposition $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ \,/\, \underset{\leftarrow}{\Omega}_R$ into a well-formed proposition B^- by nondeterministically abstracting one-by-one from either the left or right contexts.[21] This proposition B^- is semantically equivalent to the quasi-proposition $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ \,/\, \underset{\leftarrow}{\Omega}_R$ in the sense that the two intuitively satisfy the "same" left-focus judgments: We would expect the quasi-proposition to satisfy $\underset{\rightarrow}{\Omega}_L\,[\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ \,/\, \underset{\leftarrow}{\Omega}_R]\,\underset{\leftarrow}{\Omega}_R \Vdash A^+$,

[21] This procedure could be made deterministic by preferring one side over the other, but we refrain from doing so because the choice of side to prefer is completely arbitrary.

and indeed, when $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-$, we have $\underset{\rightarrow}{\Delta}_L\ [B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ if, and only if, $\underset{\rightarrow}{\Delta}_L = \underset{\rightarrow}{\Omega}_L$ and $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{\Omega}_R$ and $C^+ = A^+$. This is proved below as lemma 6.6.

This auxiliary judgment is defined inductively by the following rules.

$$\frac{}{(\cdot) \setminus \uparrow A^+ / (\cdot) \rightsquigarrow \uparrow A^+}\ \uparrow\textsc{el}$$

$$\frac{\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-}{(\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}) \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow \underset{\rightarrow}{a} \setminus B^-}\ \setminus\textsc{el} \qquad \frac{\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-}{\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / (\underset{\leftarrow}{a}\ \underset{\leftarrow}{\Omega}_R) \rightsquigarrow B^- / \underset{\leftarrow}{a}}\ /\textsc{el}$$

The $\setminus$EL rule states that the quasi-proposition $(\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a}) \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R$ is equivalent to $\underset{\rightarrow}{a} \setminus B^-$ if $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R$ is equivalent to B^-. Notice that the $\setminus$EL rule moves $\underset{\rightarrow}{a}$ from the right of $\underset{\rightarrow}{\Omega}_L$ to the left of B^-; this is admittedly counterintuitive, but it is closely related to the equally counterintuitive currying law for left-handed implication in ordered logic: $(B \bullet A) \setminus C \dashv\vdash A \setminus (B \setminus C)$. Symmetrically, the $/$EL rule is closely related to the currying law for right-handed implication: $C / (A \bullet B) \dashv\vdash (C / B) / A$.

This intuition is captured in the proof of the following lemma.

LEMMA 6.6. *If* $\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-$, *then* $\underset{\rightarrow}{\Delta}_L\ [B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ *if, and only if,* $\underset{\rightarrow}{\Delta}_L = \underset{\rightarrow}{\Omega}_L$ *and* $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{\Omega}_R$ *and* $C^+ = A^+$.

Proof. By induction over the structure of the given elaboration.

As an example case, consider

$$\frac{\underset{\rightarrow}{\Omega}_L \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow B^-}{(\underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a^+}) \setminus \uparrow A^+ / \underset{\leftarrow}{\Omega}_R \rightsquigarrow \underset{\rightarrow}{a^+} \setminus B^-}\ \setminus\textsc{el}\,.$$

We must show that $\underset{\rightarrow}{\Delta}_L\ [\underset{\rightarrow}{a^+} \setminus B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ if, and only if, $\underset{\rightarrow}{\Delta}_L = \underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a^+}$ and $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{\Omega}_R$ and $C^+ = A^+$. Indeed, the $\setminus$L rule is the unique rule for left-focusing on the proposition $\underset{\rightarrow}{a^+} \setminus B^-$, so $\underset{\rightarrow}{\Delta}_L\ [\underset{\rightarrow}{a^+} \setminus B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ if, and only if, $\underset{\rightarrow}{\Delta}_L = \underset{\rightarrow}{\Delta}'_L\ \underset{\rightarrow}{a^+}$ and $\underset{\rightarrow}{\Delta}'_L\ [B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ for some $\underset{\rightarrow}{\Delta}'_L$. By the inductive hypothesis, we have $\underset{\rightarrow}{\Delta}'_L\ [B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ if, and only if, $\underset{\rightarrow}{\Delta}'_L = \underset{\rightarrow}{\Omega}_L$ and $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{\Omega}_R$ and $C^+ = A^+$. Putting everything together, $\underset{\rightarrow}{\Delta}_L\ [\underset{\rightarrow}{a^+} \setminus B^-]\ \underset{\leftarrow}{\Delta}_R \Vdash C^+$ if, and only if, $\underset{\rightarrow}{\Delta}_L = \underset{\rightarrow}{\Omega}_L\ \underset{\rightarrow}{a^+}$ and $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{\Omega}_R$ and $C^+ = A^+$, as required. □

THE PRINCIPAL JUDGMENT is $\theta \vdash_\Sigma \Theta \rightsquigarrow \Phi$.[22] Given a string rewriting specification (Σ, Θ) and a role assignment θ, this judgment produces formula-as-process definitions Φ that, together with θ, constitute a choreography of (Σ, Θ). In other words, when $\theta \vdash_\Sigma \Theta \rightsquigarrow \Phi$ holds, the definitions Φ are a solution to the constraints induced by axioms Θ under the role assignment θ, such that θ is a (strong) bisimulation between $\longrightarrow_\Theta$ and $\longrightarrow_\Phi$. If there is no Φ for which $\theta \vdash_\Sigma \Theta \rightsquigarrow \Phi$ holds, then the role assignment θ yields no choreography of the specification (Σ, Θ).

[22] Because the alphabet Σ never changes within a derivation, we nearly always elide it.

$$\theta \vdash \Theta \rightsquigarrow \Phi \quad \text{only if} \quad \begin{array}{ccc} w & \longrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \theta(w) & \dashrightarrow_\Phi & \theta(w') \end{array} \quad \text{and} \quad \begin{array}{ccc} w & \dashrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \vdots{\scriptstyle\theta} \\ \theta(w) & \longrightarrow_\Phi & \Omega' \end{array}$$

This principal choreographing judgment is defined by just two rules:

$$\frac{}{\theta \vdash \cdot \rightsquigarrow \cdot}$$

$$\frac{\begin{array}{c}(\theta(a) = \hat{a}) \quad \theta \vdash \Theta_0 \rightsquigarrow \Phi_0 \quad (\hat{a} \notin \operatorname{dom} \Phi_0) \\ \forall i \in \mathcal{I}: \quad \left(\theta(w_i^L) = \underset{\rightarrow}{\Omega}_i^L\right) \quad \left(\theta(w_i^R) = \underset{\leftarrow}{\Omega}_i^R\right) \quad \underset{\rightarrow}{\Omega}_i^L \setminus \uparrow\bullet\theta(w_i') \,/\, \underset{\leftarrow}{\Omega}_i^R \rightsquigarrow A_i^-\end{array}}{\theta \vdash \Theta_0, \left(w_i^L \, a \, w_i^R \longrightarrow w_i'\right)_{i \in \mathcal{I}} \rightsquigarrow \Phi_0, \left(\hat{a} \triangleq \mathbin{\&}_{i \in \mathcal{I}} A_i^-\right)}$$

The first of these rules is straightforward: an empty set of string rewriting axioms choreographs as an empty set of coinductive formula-as-process definitions. The second rule is quite a lot to parse and needs to be broken down step by step:

1. Choose a symbol a that is mapped by θ to a coinductively defined proposition, $\hat{a}$. Then reorganize the axioms Θ, collecting together all axioms in Θ that have an a in their premises. Let $\left(w_i^L \, a \, w_i^R \longrightarrow w_i'\right)_{i \in \mathcal{I}}$ be those axioms, so that $\Theta = \Theta_0, \left(w_i^L \, a \, w_i^R \longrightarrow w_i'\right)_{i \in \mathcal{I}}$ for some Θ_0.

2. Inductively construct definitions Φ_0 from Θ_0 and θ, using the judgment $\theta \vdash \Theta_0 \rightsquigarrow \Phi_0$. Check that Φ_0 gives no definition for $\hat{a}$, otherwise there is some axiom in Θ_0 that contains a in its premise and $\left(w_i^L \, a \, w_i^R \longrightarrow w_i'\right)_{i \in \mathcal{I}}$ does not correctly constitute all such axioms.

3. Check, using the side condition $\theta(w_i^L) = \underset{\rightarrow}{\Omega}_i^L$, that each w_i^L contains only those symbols that map to right-directed atoms. Symmetrically, check, using the side condition $\theta(w_i^R) = \underset{\leftarrow}{\Omega}_i^R$, that each w_i^R contains only symbols that map to left-directed atoms.

4. Elaborate each quasi-proposition $\underset{\rightarrow}{\Omega}_i^L \setminus \uparrow\bullet\theta(w_i') \,/\, \underset{\leftarrow}{\Omega}_i^R$ into a semantically equivalent proposition A_i^-. Based on lemma 6.6, the left-focus judgment $\theta(w_i^L) \, [A_i^-] \, \theta(w_i^R) \Vdash \bullet\theta(w_i')$ holds, and so this proposition acts as the image of the axiom $w_i^L \, a \, w_i^R \longrightarrow w_i'$ under the role assignment θ – that is, $\theta(w_i^L) \, A_i^- \, \theta(w_i^R) \longrightarrow \theta(w_i')$.

5. Collect the A_i^-s into a single definition, $\hat{a} \triangleq \mathbin{\&}_{i \in \mathcal{I}} A_i^-$, which, based on steps 2, 3, and 4, describes all of the axioms from Θ that contain a in their premises – that is, $\theta(w_i^L) \, \hat{a} \, \theta(w_i^R) \longrightarrow_{\{\hat{a} \triangleq \mathbin{\&}_{i \in \mathcal{I}} A_i^-\}} \theta(w_i')$.

We shall now prove that this judgment produces definitions Φ that constitute a formula-as-process choreography (θ, Φ) of the string rewriting specification (Σ, Θ) – that is, that $\theta \vdash \Theta \rightsquigarrow \Phi$ implies that θ is a (strong) bisimulation between $\longrightarrow_\Theta$ and $\longrightarrow_\Phi$. As previously mentioned, because θ is injective, this amounts to proving that

$$\theta \vdash \Theta \rightsquigarrow \Phi \quad \text{only if} \quad \begin{array}{ccc} w & \longrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \downarrow{\scriptstyle\theta} \\ \theta(w) & \dashrightarrow_\Phi & \theta(w') \end{array} \quad \text{and} \quad \begin{array}{ccc} w & \dashrightarrow_\Theta & w' \\ {\scriptstyle\theta}\downarrow & & \vdots{\scriptstyle\theta} \\ \theta(w) & \longrightarrow_\Phi & \Omega' \end{array}.$$

We prove the first diagram as the following completeness theorem and then prove a stronger soundness theorem that implies the second diagram.

LEMMA 6.7 (Definition weakening). *If* $\Omega \longrightarrow_\Phi \Omega'$ *and* $\operatorname{dom}\Phi \cap \operatorname{dom}\Phi' = \emptyset$, *then* $\Omega \longrightarrow_{\Phi,\Phi'} \Omega'$.

Proof. By induction over the structure of the given rewriting step. □

THEOREM 6.8. *If* $\theta \vdash \Theta \rightsquigarrow \Phi$, *then* $w \longrightarrow_\Theta w'$ *implies* $\theta(w) \longrightarrow_\Phi \theta(w')$.

$$\begin{array}{c}\theta \vdash \Theta \rightsquigarrow \Phi \\ \text{only if} \\ \begin{array}{ccc} w & \longrightarrow_\Theta & w' \\ {\scriptstyle\theta}\big| & & \big|{\scriptstyle\theta} \\ \theta(w) & \dashrightarrow_\Phi & \theta(w') \end{array}\end{array}$$

Proof. By simultaneous structural induction on the given choreographing derivation, $\theta \vdash \Theta \rightsquigarrow \Phi$, and ordered rewriting step, $w \longrightarrow_\Theta w'$.

- Consider the case in which

$$\theta \vdash \Theta \rightsquigarrow \Phi \qquad \text{and} \qquad w = \frac{w_0 \longrightarrow_\Theta w_0'}{w_1\, w_0\, w_2 \longrightarrow_\Theta w_1\, w_0'\, w_2}\ \longrightarrow\!\text{C} = w'\,.$$

  By the inductive hypothesis, $\theta(w_0) \longrightarrow_\Phi \theta(w_0')$. It follows from ordered rewriting's $\longrightarrow$C rule that

$$\theta(w) = \theta(w_1)\,\theta(w_0)\,\theta(w_2) \longrightarrow_\Phi \theta(w_1)\,\theta(w_0')\,\theta(w_2) = \theta(w')\,.$$

- Consider the case in which

$$\frac{\begin{array}{c}(\theta(a) = \hat{a}) \quad \theta \vdash \Theta_0 \rightsquigarrow \Phi_0 \quad (\hat{a} \notin \operatorname{dom}\Phi_0) \\ \forall i \in \mathcal{I}: \quad \left(\theta(w_i^L) = \underset{\rightarrow}{\Omega}_i^L\right) \quad \left(\theta(w_i^R) = \underset{\leftarrow}{\Omega}_i^R\right) \quad \underset{\rightarrow}{\Omega}_i^L \setminus \uparrow\!\bullet\theta(w_i') \,/\, \underset{\leftarrow}{\Omega}_i^R \rightsquigarrow A_i^-\end{array}}{\theta \vdash \Theta_0, \left(w_i^L\, a\, w_i^R \longrightarrow w_i'\right)_{i\in\mathcal{I}} \rightsquigarrow \Phi_0, \left(\hat{a} \triangleq \&_{i\in\mathcal{I}}\, A_i^-\right)}$$

  and

$$w = \frac{(w_k^L\, a\, w_k^R \longrightarrow w_k') \in \Theta}{w_k^L\, a\, w_k^R \longrightarrow_\Theta w_k'}\ \longrightarrow\!\text{AX} = w'$$

  for some $k \in \mathcal{I}$, where the axioms are $\Theta = \Theta_0, (w_i^L\, a\, w_i^R \longrightarrow w_i')_{i\in\mathcal{I}}$, and the definitions are $\Phi = \Phi_0, (\&_{i\in\mathcal{I}}\, A_i^-)$.

  By lemma 6.6, $\theta(w_k^L)\ [A_k^-]\ \theta(w_k^R) \Vdash \bullet\theta(w_k')$ holds. Appending a $\&$L rule, we have $\theta(w_k^L)\ [\&_{i\in\mathcal{I}}\, A_i^-]\ \theta(w_k^R) \Vdash \bullet\theta(w_k')$. Because $[\bullet\theta(w_k')] \dashv\Vdash \theta(w_k')$, it follows by the $\longrightarrow$I rule that $\theta(w_k^L)\ \left(\&_{i\in\mathcal{I}}\, A_i^-\right)\ \theta(w_k^R) \longrightarrow_\Phi \theta(w_k')$, and so $\theta(w) = \theta(w_k^L)\ \hat{a}\ \theta(w_k^R) \longrightarrow_\Phi \theta(w_k') = \theta(w')$.

- Consider the case in which

$$\frac{\begin{array}{c}(\theta(a) = \hat{a}) \quad \theta \vdash \Theta_0 \rightsquigarrow \Phi_0 \quad (\hat{a} \notin \operatorname{dom}\Phi_0) \\ \forall i \in \mathcal{I}: \quad \left(\theta(v_i^L) = \underset{\rightarrow}{\Omega}_i^L\right) \quad \left(\theta(v_i^R) = \underset{\leftarrow}{\Omega}_i^R\right) \quad \underset{\rightarrow}{\Omega}_i^L \setminus \uparrow\!\bullet\theta(v_i') \,/\, \underset{\leftarrow}{\Omega}_i^R \rightsquigarrow A_i^-\end{array}}{\theta \vdash \Theta_0, \left(v_i^L\, a\, v_i^R \longrightarrow v_i'\right)_{i\in\mathcal{I}} \rightsquigarrow \Phi_0, \left(\hat{a} \triangleq \&_{i\in\mathcal{I}}\, A_i^-\right)}$$

  and

$$\frac{(w \longrightarrow w') \in \Theta_0}{w \longrightarrow_\Theta w'}\ \longrightarrow\!\text{AX}$$

  where $(w \longrightarrow w') \in \Theta_0$; the axioms are $\Theta = \Theta_0, (v_i^L\, a\, v_i^R \longrightarrow v_i')_{i\in\mathcal{I}}$; and the definitions are $\Phi = \Phi_0, (\&_{i\in\mathcal{I}}\, A_i^-)$.

  By the inductive hypothesis, $\theta(w) \longrightarrow_{\Phi_0} \theta(w')$. It follows from weakening (lemma 6.7) that $\theta(w) \longrightarrow_\Phi \theta(w')$.

- The case in which

$$\frac{}{\theta \vdash \cdot \rightsquigarrow \cdot} \qquad \text{and} \qquad \frac{(w \longrightarrow w') \in \Theta}{w \longrightarrow_\Theta w'} \longrightarrow\text{AX}$$

  where $\Theta = \cdot$ and $\Phi = \cdot$ is vacuous. □

Lemma 6.9. *If $\theta \vdash \Theta \rightsquigarrow \Phi$ and $\underset{\rightarrow}{\Omega}_L\, [\hat{a}]\, \underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+$, then there exists an axiom $(w_1\, a\, w_2 \longrightarrow w') \in \Theta$ such that $\underset{\rightarrow}{\Omega}_L = \theta(w_1)$, $\underset{\leftarrow}{\Omega}_R = \theta(w_2)$, and $C^+ = \bullet\theta(w')$.*

Proof. By induction over the structure of the given choreographing derivation, $\theta \vdash \Theta \rightsquigarrow \Phi$.

- Consider the case in which

$$\frac{\begin{array}{c}\theta \vdash \Theta_0 \rightsquigarrow \Phi_0 \quad (\theta(a) = \hat{a}) \quad (\hat{a} \notin \operatorname{dom}\Phi_0) \\ \forall i \in \mathcal{I} : \quad \left(\theta(w_i^L) = \underset{\rightarrow}{\Delta}_i^L\right) \quad \left(\theta(w_i^R) = \underset{\leftarrow}{\Delta}_i^R\right) \quad \underset{\rightarrow}{\Delta}_i^L \setminus \uparrow\bullet\theta(w_i') \,/\, \underset{\leftarrow}{\Delta}_i^R \rightsquigarrow A_i^-\end{array}}{\theta \vdash \Theta_0, \left(w_i^L\, a\, w_i^R \longrightarrow w_i'\right)_{i\in\mathcal{I}} \rightsquigarrow \Phi_0, \left(\hat{a} \triangleq \&_{i\in\mathcal{I}}\, A_i^-\right)}$$

  and

$$\underset{\rightarrow}{\Omega}_L\, [\hat{a} = \&_{i\in\mathcal{I}}\, A_i^-]\, \underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+$$

  where $\Theta = \Theta_0, \left(w_i^L\, a\, w_i^R \longrightarrow w_i'\right)_{i\in\mathcal{I}}$ and $\Phi = \Phi_0, \left(\hat{a} \triangleq \&_{i\in\mathcal{I}}\, A_i^-\right)$.

  By inversion on the left-focus derivation, either: $\underset{\rightarrow}{\Omega}_L\, [A_k^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+$ for some $k \in \mathcal{I}$; or $\mathcal{I}$ is empty.

  - If $\underset{\rightarrow}{\Omega}_L\, [A_k^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+$ for some $k \in \mathcal{I}$, then lemma 6.6 allows us to conclude that $\underset{\rightarrow}{\Omega}_L = \underset{\rightarrow}{\Delta}_k^L = \theta(w_k^L)$ and $\underset{\leftarrow}{\Omega}_R = \underset{\leftarrow}{\Delta}_k^R = \theta(w_k^R)$ and $C^+ = \bullet\theta(w_k')$. Also, the axiom $w_k^L\, a\, w_k^R \longrightarrow w_k'$ is contained in Θ.
  - Otherwise, if $\mathcal{I}$ is empty, then $\&_{i\in\mathcal{I}}\, A_i^- = \top$. There is no $\top$L rule to derive $\underset{\rightarrow}{\Omega}_L\, [\hat{a} = \top]\, \underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+$, so this case is vacuous.

- Consider the case in which

$$\frac{\begin{array}{c}\theta \vdash \Theta_0 \rightsquigarrow \Phi_0 \quad (\theta(b) = \hat{b}) \quad (\hat{b} \notin \operatorname{dom}\Phi_0) \\ \forall i \in \mathcal{I} : \quad \left(\theta(v_i^L) = \underset{\rightarrow}{\Delta}_i^L\right) \quad \left(\theta(v_i^R) = \underset{\leftarrow}{\Delta}_i^R\right) \quad \underset{\rightarrow}{\Delta}_i^L \setminus \uparrow\bullet\theta(v_i') \,/\, \underset{\leftarrow}{\Delta}_i^R \rightsquigarrow B_i^-\end{array}}{\theta \vdash \Theta_0, \left(v_i^L\, b\, v_i^R \longrightarrow v_i'\right)_{i\in\mathcal{I}} \rightsquigarrow \Phi_0, \left(\hat{b} \triangleq \&_{i\in\mathcal{I}}\, B_i^-\right)}$$

  and

$$\underset{\rightarrow}{\Omega}_L\, [\hat{a}]\, \underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+$$

  where $a \neq b$ and $\Theta = \Theta_0, \left(v_i^L\, b\, v_i^R \longrightarrow v_i'\right)_{i\in\mathcal{I}}$ and $\Phi = \Phi_0, \left(\hat{b} \triangleq \&_{i\in\mathcal{I}}\, B_i^-\right)$.

  By the inductive hypothesis, there exists a string rewriting axiom $(w_1\, a\, w_2 \longrightarrow w') \in \Theta_0$ such that $\underset{\rightarrow}{\Omega}_L = \theta(w_1)$ and $\underset{\leftarrow}{\Omega}_R = \theta(w_2)$ and $C^+ = \bullet\theta(w')$. The same axiom is contained in the signature Θ.

- The case in which

$$\frac{}{\theta \vdash \cdot \rightsquigarrow \cdot} \qquad \text{and} \qquad \underset{\rightarrow}{\Omega}_L\, [\hat{a}]\, \underset{\leftarrow}{\Omega}_R \Vdash_\Phi C^+$$

  where $\Theta = \cdot$ and $\Phi = \cdot$ is vacuous because there is no definition for $\hat{a}$ in the signature Φ. □

THEOREM 6.10. *If* $\theta \vdash \Theta \rightsquigarrow \Phi$ *and* $\theta(a) = \hat{a}$ *and* $\Omega_L\, \hat{a}\, \Omega_R \longrightarrow_\Phi \Omega'$, *then either:*

- $\Omega_L = \Omega'_L\, \theta(w_1)$ *and* $\Omega_R = \theta(w_2)\, \Omega'_R$ *and* $\Omega' = \Omega'_L\, \theta(w')\, \Omega'_R$ *for some contexts* Ω'_L *and* Ω'_R *and strings* w_1, w_2, *and* w' *such that* $(w_1\, a\, w_2 \longrightarrow w') \in \Theta$ *and* $\theta(w_1)\ [\hat{a}]\ \theta(w_2) \Vdash \bullet\theta(w')$;
- $\Omega_L \longrightarrow_\Phi \Omega'_L$ *for some context* Ω'_L *such that* $\Omega' = \Omega'_L\, \hat{a}\, \Omega_R$; *or*
- $\Omega_R \longrightarrow_\Phi \Omega'_R$ *for some context* Ω'_R *such that* $\Omega' = \Omega_L\, \hat{a}\, \Omega'_R$.

Proof. As a negative proposition, $\hat{a}$ serves as a barrier for interactions between Ω_L and Ω_R – in process-as-formula focused ordered rewriting, implications cannot consume negative propositions. Thus, any reduction on $\Omega_L\, \hat{a}\, \Omega_R$ must occur within either Ω_L or Ω_R alone or must arise from $\hat{a}$.

If the reduction on $\Omega_L\, \hat{a}\, \Omega_R$ arises from $\hat{a}$, then it arises from a bipole that begins by focusing on $\hat{a}$. In other words, $\Omega_L = \Omega'_L\, \underrightarrow{\Delta}_L$ and $\Omega_R = \underleftarrow{\Delta}_R\, \Omega'_R$ and $\Omega' = \Omega'_L\, \Delta'\, \Omega'_R$ for some contexts $\underrightarrow{\Delta}_L$, $\underleftarrow{\Delta}_R$, and Δ' and positive proposition C^+ such that $\underrightarrow{\Delta}_L\ [\hat{a}]\ \underleftarrow{\Delta}_R \Vdash C^+$ and $[C^+] \dashv\Vdash \Delta'$. By lemma 6.9, there exists an axiom $(w_1\, a\, w_2 \longrightarrow w') \in \Theta$ such that $\underrightarrow{\Delta}_L = \theta(w_1)$ and $\underleftarrow{\Delta}_R = \theta(w_2)$ and $C^+ = \bullet\theta(w')$. It follows that $\Delta' = \theta(w')$. □

COROLLARY 6.11 (Choreography adequacy). *If* $\theta \vdash \Theta \rightsquigarrow \Phi$, *then* $\theta(w) \longrightarrow_\Phi \Omega'$ *if, and only if, there exists* w' *such that* $w \longrightarrow_\Phi w'$ *and* $\theta(w') = \Omega'$.

6.4 *Example: Choreographing binary counters*

In this section, we revisit binary counters, *i.e.*, binary representations of natural numbers equipped with increment and decrement operations. Here we use them as an extended example of choreographing string rewriting specifications.

Recall from section 4.3 a string rewriting specification (Σ, Θ) of binary counters where the alphabet Σ and the axioms Θ are:

$$\begin{aligned}\Sigma &= \{e, b_0, b_1, i, d, z, s, b'_0\}\\ \Theta &= (e\, i \longrightarrow e\, b_1)\,,\ (b_0\, i \longrightarrow b_1)\,,\ (b_1\, i \longrightarrow i\, b_0)\,,\\ &\quad\ (e\, d \longrightarrow z)\,,\quad (b_0\, d \longrightarrow b'_0)\,,\ (b_1\, d \longrightarrow b_0\, s)\,,\\ &\quad\ (z\, b'_0 \longrightarrow z)\,,\ \ (s\, b'_0 \longrightarrow b_1\, s)\end{aligned}$$

We will present several distinct choreographies of this specification, including an object-oriented choreography that treats the increment and decrement operations as messages, and a functional choreography that instead treats those operations as processes.

6.4.1 *An object-oriented choreography*

Let θ be the role assignment that maps the bits e, b_0, and b_1 to coinductively defined processes $\hat{e}$, $\hat{b}_0$, and $\hat{b}_1$; increments i and decrements d to left-directed

$$\begin{aligned}\theta = \{&e \mapsto \hat{e}, b_0 \mapsto \hat{b}_0, b_1 \mapsto \hat{b}_1,\\ &i \mapsto \underrightarrow{i}, d \mapsto \underrightarrow{d},\\ &z \mapsto \underrightarrow{z}, s \mapsto \underrightarrow{s}, b'_0 \mapsto \hat{b}'_0\}\end{aligned}$$

Axioms, Θ	*Rewriting constraints on* Φ	*Solution,* Φ
$e\,i \longrightarrow e\,b_1$ and $e\,d \longrightarrow z$	$\hat{e}\,\underset{\leftarrow}{i} \dashrightarrow_\Phi \hat{e}\,\hat{b}_1$ and $\hat{e}\,\underset{\leftarrow}{d} \dashrightarrow_\Phi \underset{\rightarrow}{z}$	$\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d})$
$b_0\,i \longrightarrow b_1$ and $b_0\,d \longrightarrow d\,b_0'$	$\hat{b}_0\,\underset{\leftarrow}{i} \dashrightarrow_\Phi \hat{b}_1$ and $\hat{b}_0\,\underset{\leftarrow}{d} \dashrightarrow_\Phi \underset{\leftarrow}{d}\,\hat{b}_0'$	$\hat{b}_0 \triangleq (\uparrow\downarrow \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet \hat{b}_0' \,/\, \underset{\leftarrow}{d})$
$b_1\,i \longrightarrow i\,b_0$ and $b_1\,d \longrightarrow b_0\,s$	$\hat{b}_1\,\underset{\leftarrow}{i} \dashrightarrow_\Phi \underset{\leftarrow}{i}\,\hat{b}_0$ and $\hat{b}_1\,\underset{\leftarrow}{d} \dashrightarrow_\Phi \hat{b}_0\,\underset{\rightarrow}{s}$	$\hat{b}_1 \triangleq (\underset{\leftarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\leftarrow}{i}) \,\&\, (\hat{b}_0 \bullet \underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{d})$
$z\,b_0' \longrightarrow z$ and $s\,b_0' \longrightarrow b_1\,s$	$\underset{\rightarrow}{z}\,\hat{b}_0' \dashrightarrow_\Phi \underset{\rightarrow}{z}$ and $\underset{\rightarrow}{s}\,\hat{b}_0' \dashrightarrow_\Phi \hat{b}_1\,\underset{\rightarrow}{s}$	$\hat{b}_0' \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})$

Table 6.5: Deriving an object-oriented choreography of binary counters

messages $\underset{\leftarrow}{i}$ and $\underset{\leftarrow}{d}$; unary constructors z and s to right-directed messages $\underset{\rightarrow}{z}$ and $\underset{\rightarrow}{s}$; and b_0' to coinductively defined process $\hat{b}_0'$.

Two axioms in Θ mention e in their premises: $e\,i \longrightarrow e\,b_1$ and $e\,d \longrightarrow z$. Under the role assignment θ, these axioms induce the rewritings

$$\hat{e}\,\underset{\leftarrow}{i} \dashrightarrow_\Phi \hat{e}\,\hat{b}_1 \qquad \text{and} \qquad \hat{e}\,\underset{\leftarrow}{d} \dashrightarrow_\Phi \underset{\rightarrow}{z}$$

as constraints on Φ that must be satisfied if (θ, Φ) is to be a meaningful choreography of the binary counter specification. Solving these for $\hat{e}$, we obtain the definition

$$\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d})\,.$$

Similar reasoning allows us to construct coinductive definitions for $\hat{b}_0$, $\hat{b}_1$, and $\hat{b}_0'$ as the solutions of the other constraints induced from the axioms Θ by θ. (See table 6.5 for a sketch.) In full, the solution to these constraints is the definitions Φ:

$$\begin{aligned}\Phi = \big(&\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d})\big)\,,\\ \big(&\hat{b}_0 \triangleq (\uparrow\downarrow \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet \hat{b}_0' \,/\, \underset{\leftarrow}{d})\big)\,,\\ \big(&\hat{b}_1 \triangleq (\underset{\leftarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\leftarrow}{i}) \,\&\, (\hat{b}_0 \bullet \underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{d})\big)\,,\\ \big(&\hat{b}_0' \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})\big)\,.\end{aligned}$$

In other words, under the role assignment θ, the string rewriting axioms for the binary counter are choreographed to the coinductive propositions defined in Φ. It is easy, if tedious, to verify that the formal construction described in section generates the same definitions, Φ:

Proposition 6.12. *For the above string rewriting specification* (Σ, Θ) *and role assignment* θ*, the judgment* $\theta \vdash \Theta \rightsquigarrow \Phi$ *holds.*

This choreography might be called *object-oriented* for its similarity to the eponymous programming paradigm. In that paradigm, computation is centered around message exchange between stateful objects – data are stored by objects, and those data are manipulated by exchanging messages with the relevant objects.

This choreography of the binary counter specification behaves similarly:[23] its data – the bits e, b_0, and b_1 – are represented as processes, and its operations – the increments i and decrements d – are represented as messages that the processes dispatch on. For example, $\hat{e}$ is the coinductively defined

[23] For a study of the relationship between (session-typed) processes and objects, see Balzer and Pfenning (2015).

process that waits to receive either the increment message $\underset{\leftarrow}{i}$ or the decrement message $\underset{\leftarrow}{d}$ from its right-hand neighbor. If $\underset{\leftarrow}{i}$ is received, then $\hat{e}$ spawns a new process, $\hat{b}_1$, to its right and then continues recursively as $\hat{e}$. Otherwise, if $\underset{\leftarrow}{d}$ is received, then $\hat{e}$ sends the message $\underset{\rightarrow}{z}$ as a response.

Recall from section 4.3 that string rewriting specifications of binary counters were assigned natural number denotations according to the relations $\approx_{\text{V}}$, $\approx_{\text{I}}$, and $\approx_{\text{D}}$. Based on the role assignment that underlies this choreography, we can lift these denotations to choreographed counters: For instance, Ω is an increment context that denotes n exactly when $\theta^{-1}(\Omega) \approx_{\text{I}} n$; and so $\hat{e}\,\underset{\leftarrow}{i}\,\hat{b}_1$ denotes 3 because $\theta^{-1}(\hat{e}\,\underset{\leftarrow}{i}\,\hat{b}_1) = e\,i\,b_1 \approx_{\text{I}} 3$. We could even assign denotations directly to choreographed contexts by defining new $\approx_{\text{V}}$, $\approx_{\text{I}}$, and $\approx_{\text{D}}$ relations on choreographed contexts.

$$\frac{}{\hat{e} \approx_{\text{V}} 0}\ \hat{e}\text{-V} \qquad \frac{\Omega \approx_{\text{V}} n}{\Omega\,\hat{b}_0 \approx_{\text{V}} 2n}\ \hat{b}_0\text{-V} \qquad \frac{\Omega \approx_{\text{V}} n}{\Omega\,\hat{b}_1 \approx_{\text{V}} 2n+1}\ \hat{b}_1\text{-V}$$

$$\frac{}{\hat{e} \approx_{\text{I}} 0}\ \hat{e}\text{-I} \qquad \frac{\Omega \approx_{\text{I}} n}{\Omega\,\hat{b}_0 \approx_{\text{I}} 2n}\ \hat{b}_0\text{-I} \qquad \frac{\Omega \approx_{\text{I}} n}{\Omega\,\hat{b}_1 \approx_{\text{I}} 2n+1}\ \hat{b}_1\text{-I} \qquad \frac{\Omega \approx_{\text{I}} n}{\Omega\,\underset{\leftarrow}{i} \approx_{\text{I}} n+1}\ \underset{\leftarrow}{i}\text{-I}$$

$$\frac{\Omega \approx_{\text{I}} n}{\Omega\,\underset{\leftarrow}{d} \approx_{\text{D}} n}\ \underset{\leftarrow}{d}\text{-D} \qquad \frac{}{\underset{\rightarrow}{z} \approx_{\text{D}} 0}\ \underset{\rightarrow}{z}\text{-D} \qquad \frac{\Omega \approx_{\text{I}} n}{\Omega\,\underset{\rightarrow}{s} \approx_{\text{D}} n+1}\ \underset{\rightarrow}{s}\text{-D} \qquad \frac{\Omega \approx_{\text{D}} n}{\Omega\,\hat{b}'_0 \approx_{\text{D}} 2n}\ \hat{b}'_0\text{-D}$$

We will say that a context Ω is an *increment counter* or *increment context* if $\Omega \approx_{\text{I}} n$ for some natural number n; likewise, we will say that a context Ω is an *decrement counter* or *decrement context* if $\Omega \approx_{\text{D}} n$ for some n.

Theorem 6.13. *The following hold for all Ω and n.*

- *$\Omega \approx_{\text{V}} n$ if, and only if, $\Omega = \theta(w)$ for some w such that $w \approx_{\text{V}} n$.*
- *$\Omega \approx_{\text{I}} n$ if, and only if, $\Omega = \theta(w)$ for some w such that $w \approx_{\text{I}} n$.*
- *$\Omega \approx_{\text{D}} n$ if, and only if, $\Omega = \theta(w)$ for some w such that $w \approx_{\text{D}} n$.*

Proof. In each direction, by structural induction on the derivation of the denotation. □

Just as we proved the string rewriting specification of binary counters to be adequate with respect to the natural number denotation, we can also show this object-oriented choreography to be adequate. What is interesting is that the adequacy of this choreography comes for nearly free – it can be established by composing the string rewriting specification's adequacy theorem with the theorems that show an arbitrary choreography to be sound and complete with respect to its underlying string rewriting specification.

Recall from section 4.3 that the string rewriting specification adequately describes increments and decrements:

Theorem 4.4 (Small-step adequacy of increments).
Value soundness *If $w \approx_{\text{V}} n$, then $w \approx_{\text{I}} n$ and $w \not\longrightarrow$.*
Preservation *If $w \approx_{\text{I}} n$ and $w \longrightarrow w'$, then $w' \approx_{\text{I}} n$.*

Progress *If $w \approx_{\mathrm{I}} n$, then either: $w \longrightarrow w'$ for some w'; or $w \approx_{\mathrm{V}} n$.*
Termination *If $w \approx_{\mathrm{I}} n$, then every rewriting sequence from w is finite.*

Theorem 4.6 (Small-step adequacy of decrements).
Preservation *If $w \approx_{\mathrm{D}} n$ and $w \longrightarrow w'$, then $w' \approx_{\mathrm{D}} n$.*
Progress *If $w \approx_{\mathrm{D}} n$, then either:*
- *$w \longrightarrow w'$, for some w';*
- *$n = 0$ and $w = z$; or*
- *$n > 0$ and $w = w'\,s$, for some w' such that $w' \approx_{\mathrm{I}} n - 1$.*

Termination *If $w \approx_{\mathrm{D}} n$, then every rewriting sequence from w is finite.*

Corollary 4.7 (Big-step adequacy of decrements). *If $w \approx_{\mathrm{D}} n$, then:*
- *$w \Longrightarrow z$ if, and only if, $n = 0$;*
- *$w \Longrightarrow w'\,s$ for some w' such that $w' \approx_{\mathrm{I}} n - 1$, if $n > 0$; and*
- *$w \Longrightarrow w'\,s$ only if $n > 0$ and $w' \approx_{\mathrm{I}} n - 1$.*

Combining these theorems with theorem 6.8, we have the immediate corollary:

Corollary 6.14 (Adequacy of object-oriented choreography). *The following hold.*

Preservation *If $\Omega \approx_{\mathrm{I}} n$ and $\Omega \longrightarrow_{\Phi} \Omega'$, then $\Omega' \approx_{\mathrm{I}} n$. If $\Omega \approx_{\mathrm{D}} n$ and $\Omega \longrightarrow_{\Phi} \Omega'$, then $\Omega' \approx_{\mathrm{D}} n$.*

Big-step *If $\Omega \approx_{\mathrm{D}} n$, then:*
- *$\Omega \Longrightarrow_{\Phi} \underrightarrow{z}$ if, and only if, $n = 0$;*
- *$\Omega \Longrightarrow_{\Phi} \Omega' \underrightarrow{s}$ for some Ω' such that $\Omega' \approx_{\mathrm{I}} n - 1$, if $n > 0$; and*
- *$\Omega \Longrightarrow_{\Phi} \Omega' \underrightarrow{s}$ only if $n > 0$ and $\Omega' \approx_{\mathrm{I}} n - 1$.*

6.4.2 *A functional choreography*

The object-oriented choreography is not the only choreography possible for the binary counter specification, however.

Let θ' be a role assignment that is (roughly) dual to θ – that is, let θ' map the bits e, b_0, and b_1 to right-directed messages $\underrightarrow{e}$, $\underrightarrow{b}_0$, and $\underrightarrow{b}_1$; increments i and decrements d to coinductively defined processes $\hat{\imath}$ and $\hat{d}$; unary constructors z and s to right-directed messages $\underrightarrow{z}$ and $\underrightarrow{s}$; and b_0' to the coinductively defined process $\hat{b}_0'$.

$$\theta' = \{e \mapsto \underrightarrow{e}, b_0 \mapsto \underrightarrow{b}_0, b_1 \mapsto \underrightarrow{b}_1, \\ i \mapsto \hat{\imath}, d \mapsto \hat{d}, \\ z \mapsto \underrightarrow{z}, s \mapsto \underrightarrow{s}, b_0' \mapsto \hat{b}_0'\}$$

Once again, we can construct a choreography from the string rewriting axioms Θ by solving constraints in the form of rewritings. Three axioms from Θ mention i in their premises: $e\,i \longrightarrow e\,b_1$, $b_0\,i \longrightarrow b_1$, and $b_1\,i \longrightarrow i\,b_0$. Under the role assignment θ', these axioms induce the rewritings

$$\underrightarrow{e}\,\hat{\imath} \dashrightarrow_{\Phi'} \underrightarrow{e}\,\underrightarrow{b}_1 \qquad \text{and} \qquad \underrightarrow{b}_0\,\hat{\imath} \dashrightarrow_{\Phi'} \underrightarrow{b}_1 \qquad \text{and} \qquad \underrightarrow{b}_1\,\hat{\imath} \dashrightarrow_{\Phi'} \hat{\imath}\,\underrightarrow{b}_0$$

as constraints on Φ' that must be satisfied if (θ', Φ') is to be a choreography of the binary counter specification. Solving these constraints for $\hat{\imath}$, we obtain the definition

$$\hat{\imath} \triangleq (\underrightarrow{e} \setminus \underrightarrow{e} \bullet \underrightarrow{b}_1) \mathbin{\&} (\underrightarrow{b}_0 \setminus \underrightarrow{b}_1) \mathbin{\&} (\underrightarrow{b}_1 \setminus \hat{\imath} \bullet \underrightarrow{b}_0)\,.$$

Axioms, Θ	*Rewriting constraints on* Φ'	*Solution,* Φ'
$e\,i \longrightarrow e\,b_1$ and $b_0\,i \longrightarrow b_1$ and $b_1\,i \longrightarrow i\,b_0$	$\underset{\rightarrow}{e}\,\hat{\imath} \dashrightarrow_{\Phi'} \underset{\rightarrow}{e}\,\underset{\rightarrow}{b}_1$ and $\underset{\rightarrow}{b}_0\,\hat{\imath} \dashrightarrow_{\Phi'} \underset{\rightarrow}{b}_1$ and $\underset{\rightarrow}{b}_1\,\hat{\imath} \dashrightarrow_{\Phi'} \hat{\imath}\,\underset{\rightarrow}{b}_0$	$\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \hat{\imath} \bullet \underset{\rightarrow}{b}_0)$
$e\,d \longrightarrow z$ and $b_0\,d \longrightarrow d\,b'_0$ and $b_1\,d \longrightarrow b_0\,s$	$\underset{\rightarrow}{e}\,\hat{d} \dashrightarrow_{\Phi'} \underset{\rightarrow}{z}$ and $\underset{\rightarrow}{b}_0\,\hat{d} \dashrightarrow_{\Phi'} \hat{d}\,\hat{b}'_0$ and $\underset{\rightarrow}{b}_1\,\hat{d} \dashrightarrow_{\Phi'} \underset{\rightarrow}{b}_0\,\underset{\rightarrow}{s}$	$\hat{d} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \hat{d} \bullet \hat{b}'_0) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \underset{\rightarrow}{b}_0 \bullet \underset{\rightarrow}{s})$
$z\,b'_0 \longrightarrow z$ and $s\,b'_0 \longrightarrow b_1\,s$	$\underset{\rightarrow}{z}\,\hat{b}'_0 \dashrightarrow_{\Phi'} \underset{\rightarrow}{z}$ and $\underset{\rightarrow}{s}\,\hat{b}'_0 \dashrightarrow_{\Phi'} \hat{b}_1\,\underset{\rightarrow}{s}$	$\hat{b}'_0 \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \underset{\rightarrow}{b}_1 \bullet \underset{\rightarrow}{s})$

Table 6.6: Deriving a functional choreography of binary counters

Upon solving the remaining constraints for the other coinductively defined propositions, $\hat{d}$ and $\hat{b}'_0$ (table 6.6),we arrive at the definitions

$$\begin{aligned}\Phi' = &\left(\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \hat{\imath} \bullet \underset{\rightarrow}{b}_0)\right),\\ &\left(\hat{d} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \hat{d} \bullet \hat{b}'_0) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \underset{\rightarrow}{b}_0 \bullet \underset{\rightarrow}{s})\right),\\ &\left(\hat{b}'_0 \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})\right).\end{aligned}$$

Again, it is easy to verify that these definitions are exactly those that are constructed by the formal description of the choreographing algorithm:

PROPOSITION 6.15. *For the above string rewriting specification* (Σ, Θ) *and role assignment* θ'*, the judgment* $\theta' \vdash \Theta \rightsquigarrow \Phi'$ *holds.*

In contrast with the previous, object-oriented choreography, this choreography treats its data – the bits e, b_0, and b_1 – as messages that are manipulated by processes that represent the operations – increments i and decrements d. For this reason, the choreography (θ', Φ') might be called *functional* for its similarity to functional programming.

6.4.3 *Duality and other choreographies*

These two (roughly) dual object-oriented and functional choreographies hint at a fundamental duality between the object-oriented and functional programming paradigms.

It is briefly tempting to think that a general duality theorem for choreographies might exist. Perhaps if (θ, Φ) is a choreography of the specification (Σ, Θ), there exists a dual choreography $(\theta^\perp, \Phi^\perp)$ in which $\theta^\perp$ maps a symbol a to a message exactly when θ mapped it to a process?

Such a theorem does not exist. As a counterexample, recall the string rewriting specification (Σ, Θ) and choreography (θ, Φ) given by

$$\begin{aligned}&\Sigma = \{a, b\}\\ &\Theta = (a\,b \longrightarrow b), (b \longrightarrow \epsilon)\end{aligned} \quad \text{and} \quad \begin{aligned}&\theta = \{a \mapsto \underset{\rightarrow}{a}, b \mapsto \hat{b}\}\\ &\Phi = \left(\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\!\downarrow \hat{b}) \,\&\, \uparrow\mathbf{1}\right).\end{aligned}$$

For this choreography, the dual role assignment, $\theta^\perp$, would map b to a message, either $\underset{\leftarrow}{b}$ or $\underset{\rightarrow}{b}$. And, the axiom $b \longrightarrow \epsilon$ would, under $\theta^\perp$, induce either $\underset{\leftarrow}{b} \dashrightarrow_{\Phi^\perp}(\cdot)$ or $\underset{\rightarrow}{b} \dashrightarrow_{\Phi^\perp}(\cdot)$ as a constraint. Neither of these possible constraints is satisfiable in the formula-as-process ordered rewriting framework

because the premises contain only messages, which are passive and have no definitions.

One might also ask if a theorem is possible if some additional conditions are imposed on the specification. For instance, at first glance, a duality theorem might seem possible for those specifications in which all axioms' premises contain exactly two symbols. Unfortunately, this is not the case. Consider, as a counterexample, the string rewriting specification (Σ, Θ) and the choreography (θ, Φ) given by

$$\Sigma = \{a, b, c\} \qquad \Theta = (a\,b \longrightarrow b)\,,\ (b\,c \longrightarrow b)$$

and

$$\theta = \{a \mapsto \underset{\rightarrow}{a}, b \mapsto \hat{b}, c \mapsto \underset{\leftarrow}{c}\}$$
$$\Phi = \left(\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\!\downarrow\hat{b}) \mathbin{\&} (\uparrow\!\downarrow\hat{b} \,/\, \underset{\leftarrow}{c})\right).$$

For this choreography, the dual role assignment, $\theta^{\perp}$, would map b to a message, either $\underset{\leftarrow}{b}$ or $\underset{\rightarrow}{b}$. But either choice leads to unsatisfiable constraints. Depending on whether $\theta^{\perp}$ maps b to $\underset{\leftarrow}{b}$ or $\underset{\rightarrow}{b}$, the induced constraints are either:

$$\hat{a}\,\underset{\leftarrow}{b} \dashrightarrow_{\Phi^{\perp}} \underset{\leftarrow}{b} \text{ and } \underset{\leftarrow}{b}\,\hat{c} \dashrightarrow_{\Phi^{\perp}} \underset{\leftarrow}{b} \qquad \text{or} \qquad \hat{a}\,\underset{\rightarrow}{b} \dashrightarrow_{\Phi^{\perp}} \underset{\rightarrow}{b} \text{ and } \underset{\rightarrow}{b}\,\hat{c} \dashrightarrow_{\Phi^{\perp}} \underset{\rightarrow}{b},$$

respectively. In either case, the constraints are unsatisfiable because one premise in each group involves a message directed outward, away from the premise's process.[24]

[24] See section 6.1.2.

Besides these object-oriented and functional choreographies, the binary counter specification has two other, related choreographies. The two alternatives are broadly similar to the object-oriented and functional choreographies, with two exceptions: the unary constructors z and s are treated as processes, not messages; and b'_0 is treated as a message, not a process. Instead of responding to a decrement with either a $\underset{\rightarrow}{z}$ or $\underset{\rightarrow}{s}$ message, these choreographies transform the binary counter into head-unary form where the head is a process – either $\hat{z}$ or $\hat{s}$.

One problem with these choreographies, however, is that, without a $\underset{\rightarrow}{z}$ or $\underset{\rightarrow}{s}$ response message, there is no way to observe the counter's state. Under these choreographies, all counters are, in some sense, equivalent because they can't be distinguished by any of the nonexistent observations. We will return to the idea of observational equivalence in the following chapter.

6.5 *Example: Choreographing nondeterministic finite automata*

Recall from section 4.2 our string rewriting specification of how an NFA processes its input. Given an NFA $\mathcal{A} = (Q, \Delta, F)$ over an input alphabet Σ, the NFA's operational semantics is adequately captured by the string rewriting specification $(\Sigma \uplus \{\$, y, n\}, \Theta)$, where the axioms Θ are given by

$$\Theta = \{a\,q \longrightarrow q'_a \mid (q'_a \in \Delta(q, a))\} \cup \{\$\,q \longrightarrow F(q) \mid q \in Q\} \quad \text{where } F(q) = \begin{cases} y & \text{if } q \in F \\ n & \text{if } q \notin F\,. \end{cases}$$

Object-oriented–like alternative	*Functional-like alternative*
$\theta^* = \theta[z \mapsto \hat{z}, s \mapsto \hat{s}, b_0' \mapsto \underset{\leftarrow}{b_0'}]$	$\theta^\dagger = \theta'[z \mapsto \hat{z}, s \mapsto \hat{s}, b_0' \mapsto \underset{\leftarrow}{b_0'}]$
$\Phi^* = \big(\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\rightarrow}{i}) \,\&\, (\uparrow\downarrow\hat{z} \,/\, \underset{\leftarrow}{d})\big),$	$\Phi^\dagger = \big(\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b_1}) \,\&\, (\underset{\rightarrow}{b_0} \setminus \underset{\rightarrow}{b_1}) \,\&\, (\underset{\rightarrow}{b_1} \setminus \hat{\imath} \bullet \underset{\rightarrow}{b_0})\big),$
$\big(\hat{b}_0 \triangleq (\uparrow\downarrow\hat{b}_1 \,/\, \underset{\rightarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet \underset{\leftarrow}{b_0'} \,/\, \underset{\leftarrow}{d})\big),$	$\big(\hat{d} \triangleq (\underset{\rightarrow}{e} \setminus \uparrow\downarrow\hat{z}) \,\&\, (\underset{\rightarrow}{b_0} \setminus \hat{d} \bullet \underset{\leftarrow}{b_0'}) \,\&\, (\underset{\rightarrow}{b_1} \setminus \underset{\rightarrow}{b_0} \bullet \hat{s})\big),$
$\big(\hat{b}_1 \triangleq (\underset{\rightarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\rightarrow}{i}) \,\&\, (\hat{b}_0 \bullet \hat{s} \,/\, \underset{\leftarrow}{d})\big),$	$\big(\hat{z} \triangleq \uparrow\downarrow\hat{z} \,/\, \underset{\leftarrow}{b_0'}\big),$
$\big(\hat{z} \triangleq \uparrow\downarrow\hat{z} \,/\, \underset{\leftarrow}{b_0'}\big),$	$\big(\hat{s} \triangleq \underset{\rightarrow}{b_1} \bullet \hat{s} \,/\, \underset{\leftarrow}{b_0'}\big)$
$\big(\hat{s} \triangleq \hat{b}_1 \bullet \hat{s} \,/\, \underset{\leftarrow}{b_0'}\big)$	

Table 6.7: Two other choreographies for the binary counter specification

As a second extended example of a choreography, we would now like to choreograph this specification in the formula-as-process ordered rewriting framework. As with the binary counter specification, there are, in fact, two distinct choreographies for this string rewriting specification of NFAs – one functional and one object-oriented.

6.5.1 *A functional choreography*

Let θ be the role assignment that maps each input symbol $a \in \Sigma$ to a right-directed message, $\underset{\rightarrow}{a}$; the end-of-word marker, \$, to a right-directed message, $\underset{\rightarrow}{\$}$; each state $q \in Q$ to a coinductively defined proposition, $\hat{q}$; and the acceptance and rejection symbols, y and n, to right-directed messages, $\underset{\rightarrow}{y}$ and $\underset{\rightarrow}{n}$. In other words, the input word is transmitted as a sequence of messages to a process $\hat{q}$ that tracks the NFA's current state.

$$\begin{aligned}\theta = {} & \{a \mapsto \underset{\rightarrow}{a} \mid a \in \Sigma\} \cup \{\$ \mapsto \underset{\rightarrow}{\$}\} \\ & \cup \{q \mapsto \hat{q} \mid q \in Q\} \\ & \cup \{y \mapsto \underset{\rightarrow}{y}, n \mapsto \underset{\rightarrow}{n}\}\end{aligned}$$

Choose an arbitrary state $q \in Q$. Under the role assignment θ, the axioms in Θ that mention q in their premises induce the rewritings

$$\begin{aligned}&\{\underset{\rightarrow}{\$}\,\hat{q} \dashrightarrow_\Phi \underset{\rightarrow}{F}(q)\} \\ &\cup \textstyle\bigcup_{a\in\Sigma}\{\underset{\rightarrow}{a}\,\hat{q} \dashrightarrow_\Phi \hat{q}_a' \mid q_a' \in \Delta(q,a)\}\end{aligned} \qquad \text{where } \underset{\rightarrow}{F}(q) = \begin{cases} \underset{\rightarrow}{y} & \text{if } q \in F \\ \underset{\rightarrow}{n} & \text{if } q \notin F \end{cases}$$

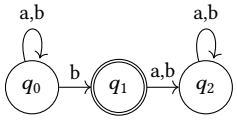
as constraints on Φ that must be satisfied if (θ, Φ) is to be a meaningful choreography of the NFA specification $(\Sigma \uplus \{\$, y, n\}, \Theta)$. Solving these constraints for $\hat{q}$, we obtain the definition

$$\hat{q} \triangleq (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{F}(q)) \,\&\, \mathop{\&}_{a\in\Sigma}\big(\underset{\rightarrow}{a} \setminus (\&_{q_a'\in\Delta(q,a)} \uparrow\downarrow\hat{q}_a')\big),$$

and therefore the full choreographing signature is

$$\Phi = \left(\hat{q} \triangleq (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{F}(q)) \,\&\, \mathop{\&}_{a\in\Sigma}\big(\underset{\rightarrow}{a} \setminus (\&_{q_a'\in\Delta(q,a)} \uparrow\downarrow\hat{q}_a')\big)\right)_{q\in Q}$$

As a concrete example, the adjacent figure recalls from fig. 2.1 an NFA that accepts those words, over the alphabet $\Sigma = \{a, b\}$, that end with b, and also gives a choreographing signature for that NFA.



$$\begin{aligned}\Phi = {} & \big(\hat{q}_0 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_0) \,\&\, (\underset{\rightarrow}{b} \setminus (\uparrow\downarrow\hat{q}_0 \,\&\, \uparrow\downarrow\hat{q}_1)) \,\&\, (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{n})\big), \\ & \big(\hat{q}_1 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_2) \,\&\, (\underset{\rightarrow}{b} \setminus \uparrow\downarrow\hat{q}_2) \,\&\, (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{y})\big), \\ & \big(\hat{q}_2 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_2) \,\&\, (\underset{\rightarrow}{b} \setminus \uparrow\downarrow\hat{q}_2) \,\&\, (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{n})\big)\end{aligned}$$

Figure 6.6: An NFA that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b; and a choreography

Similarly to one of the binary counter's choreographies, this choreography might be called 'functional' because the data, an input string, are represented

by messages that are acted on in a function-like way by the current state's process, $\hat{q}$.

Proposition 6.16. *For the above string rewriting specification* $(\Sigma \uplus \{\$, y, n\}, \Theta)$ *and role assignment* θ*, the judgment* $\theta \vdash \Theta \rightsquigarrow \Phi$ *holds (up to focusing equivalence).*

Recall from section 4.2 the adequacy theorem for the string rewriting specification of NFAs.

Theorem 4.2 (Adequacy of NFA specification). *Let* $\mathcal{A} = (Q, \Delta, F)$ *be an NFA over the input alphabet* Σ*.*

- $q \xrightarrow{a} q'_a$ *if, and only if,* $a\, q \longrightarrow q'_a$*, for all input symbols* $a \in \Sigma$*.*
- $q \in F$ *if, and only if,* $\$\, q \longrightarrow y$*.*
- $q \xrightarrow{w} q'$ *if, and only if,* $w^{\mathrm{R}}\, q \Longrightarrow q'$*, for all finite words* $w \in \Sigma^*$*.*

What we would like now is a theorem that relates an NFA not to a string rewriting specification but to the above functional choreography. In the specific instance of this functional choreography of the NFA, corollary 6.11, the adequacy of the choreographing procedure, can be reduced to the following corollary – a result that relates the string rewriting specification to the choreography.

Corollary 6.17. *For the above string rewriting specification* $(\Sigma \uplus \{\$, y, n\}, \Theta)$ *and choreography* (θ, Φ)*, the choreography is adequate with respect to the specification:*

- $a\, q \longrightarrow_\Theta q'_a$ *only if* $\underset{\rightarrow}{a}\, \hat{q} \longrightarrow_\Phi \hat{q}'_a$*. Moreover, if* $\underset{\rightarrow}{a}\, \hat{q} \longrightarrow_\Phi \Omega'$*, then* $a\, q \longrightarrow_\Theta q'_a$ *for some state* q'_a *such that* $\Omega' = \hat{q}'_a$*.*
- $\$\, q \longrightarrow_\Theta F(q)$ *if, and only if,* $\underset{\rightarrow}{\$}\, \hat{q} \longrightarrow_\Phi \underset{\rightarrow}{F}(q)$*.*
- $w^{\mathrm{R}}\, q \longrightarrow_\Theta q'$ *only if* $\underset{\rightarrow}{a}^{\mathrm{R}}\, \hat{q} \longrightarrow_\Phi \hat{q}'$*. Moreover, if* $\underset{\rightarrow}{w}^{\mathrm{R}}\, \hat{q} \longrightarrow_\Phi \Omega'$*, then* $w^{\mathrm{R}}\, q \longrightarrow_\Theta q'$ *for some state* q' *such that* $\Omega' = \hat{q}'$*.*

By composing this with theorem 4.2, the adequacy of the NFA string rewriting specification, we arrive at:

$$\underset{\rightarrow}{w}^{\mathrm{R}} = \begin{cases} (\cdot) & \text{if } w = \epsilon \\ \underset{\rightarrow}{w_0}^{\mathrm{R}}\, \underset{\rightarrow}{a} & \text{if } w = a\, w_0 \end{cases}$$

Figure 6.7: An anti-homomorphism from input words to sequences of right-directed messages. Notice that $\underset{\rightarrow}{w}^{\mathrm{R}} = \theta(w^{\mathrm{R}})$, where $^{\mathrm{R}}$ is defined in fig. 4.7.

Corollary 6.18 (Adequacy of the functional NFA choreography). *Let* $\mathcal{A} = (Q, \Delta, F)$ *be an NFA over the input alphabet* Σ*, with choreography* (θ, Φ) *as described above. The following hold.*

- $q \xrightarrow{a} q'_a$ *only if* $\underset{\rightarrow}{a}\, \hat{q} \longrightarrow_\Phi \hat{q}'_a$*. Also, if* $\underset{\rightarrow}{a}\, \hat{q} \longrightarrow_\Phi \Omega'$*, then* $q \xrightarrow{a} q'_a$ *for some state* q'_a *such that* $\Omega' = \hat{q}'_a$*.*
- $q \in F$ *if, and only if,* $\underset{\rightarrow}{\$}\, \hat{q} \longrightarrow_\Phi (\cdot)$*.*
- $q \xrightarrow{w} q'$ *only if* $\underset{\rightarrow}{w}^{\mathrm{R}}\, \hat{q} \Longrightarrow_\Phi \hat{q}'$*. Also, if* $\underset{\rightarrow}{w}^{\mathrm{R}}\, \hat{q} \Longrightarrow_\Phi \Omega'$*, then* $q \xrightarrow{w} q'$ *for some state* q' *such that* $\Omega' = \hat{q}'$*.*

This corollary gives – nearly for free – an end-to-end adequacy result for the functional NFA choreography with respect to the mathematical model of NFAs. Examining the first part, its first clause captures the completeness of the choreography: each NFA transition is simulated by a corresponding rewriting in the choreography. The second clause captures the soundness of the choreography: each rewriting simulates some NFA transition.

6.5.2 *An object-oriented choreography*

Once again, the functional choreography is not the only choreography possible for the NFA specification. As for the binary counter, there is an 'object-oriented' choreography that treats the states as messages that effect a response from processes that represent symbols of an input word. In this way, we may use a role assignment that is roughly dual to the assignment used in the preceding functional choreography.

Specifically, let θ' be the role assignment that maps each input symbol $a \in \Sigma$ and the end-of-word marker, \$, to coinductively defined propositions, $\hat{a}$ and $\hat{\$}$, respectively; each state $q \in Q$ to a left-directed message, $\underset{\leftarrow}{q}$; and the rejection symbol, n, to a right-directed message, $\underset{\rightarrow}{n}$.

$$\begin{aligned}\theta' = {} & \{a \mapsto \hat{a} \mid a \in \Sigma\} \cup \{\$ \mapsto \hat{\$}\} \\ & \cup \{q \mapsto \underset{\leftarrow}{q} \mid q \in Q\} \\ & \cup \{y \mapsto \underset{\rightarrow}{y}, n \mapsto \underset{\rightarrow}{n}\}\end{aligned}$$

Under the role assignment θ', the axioms in Θ that mention a and \$ in their premises induce the rewritings

$$\bigcup_{q \in Q} \{\hat{a}\, \underset{\leftarrow}{q} \dashrightarrow_{\Phi'} \underset{\leftarrow}{q'_a} \mid q'_a \in \Delta(q, a)\} \quad \text{and} \quad \bigcup_{q \in Q} \{\hat{\$}\, \underset{\leftarrow}{q} \dashrightarrow_{\Phi'} \underset{\rightarrow}{F(q)}\},$$

respectively, as constraints on Φ' that must be satisfied if (θ', Φ') is to be a meaningful choreography of the NFA specification. Solving these constraints for $\hat{a}$ and $\hat{\$}$, respectively, we obtain the definitions

$$\hat{a} \triangleq \bigwith_{q \in Q} \left(\&_{q'_a \in \Delta(q,a)} (\underset{\leftarrow}{q'_a} \,/\, \underset{\leftarrow}{q})\right) \quad \text{and} \quad \hat{\$} \triangleq \bigwith_{q \in Q} \left(\uparrow \underset{\rightarrow}{F(q)} \,/\, \underset{\leftarrow}{q}\right).$$

In full, the choreographing signature Φ' is therefore

$$\Phi' = \left(\hat{\$} \triangleq \&_{q \in Q} (\uparrow \underset{\rightarrow}{F(q)} \,/\, \underset{\leftarrow}{q})\right), \left(\hat{a} \triangleq \bigwith_{q \in Q} \left(\&_{q'_a \in \Delta(q,a)} (\underset{\leftarrow}{q'_a} \,/\, \underset{\leftarrow}{q})\right)\right)_{a \in \Sigma}$$

Indeed, this is the same choreographing signature that is produced by the formal procedure:

Proposition 6.19. *For the string rewriting specification* $(\Sigma \uplus \{\$, y, n\}, \Theta)$ *and role assignment* θ'*, the judgment* $\theta' \vdash \Theta \rightsquigarrow \Phi'$ *holds.*

As for the functional choreography, we may then establish a shortcut adequacy theorem for this object-oriented choreography as a corollary of earlier results. Composing this proposition with theorem 4.2, the adequacy of formula-as-process choreographies with respect to their underlying string rewriting specifications, we arrive at:

Corollary 6.20. *Let* $\mathcal{A} = (Q, \Delta, F)$ *be an NFA over the input alphabet* Σ*, with choreography* (θ', Φ') *as described above. The following hold.*

- $q \xrightarrow{a} q'_a$ *only if* $\hat{a}\, \underset{\leftarrow}{q} \longrightarrow_{\Phi'} \underset{\leftarrow}{q'_a}$*. Also, if* $\hat{a}\, \underset{\leftarrow}{q} \longrightarrow_{\Phi'} \Omega'$*, then* $q \xrightarrow{a} q'_a$ *for some state* q'_a *such that* $\Omega' = \underset{\leftarrow}{q'_a}$*.*
- $q \in F$ *if, and only if,* $\hat{\$}\, \underset{\leftarrow}{q} \longrightarrow_{\Phi'} (\cdot)$*.*
- $q \xrightarrow{w} q'$ *only if* $\hat{w}^{\mathrm{R}}\, \underset{\leftarrow}{q} \Longrightarrow_{\Phi'} \underset{\leftarrow}{q'}$*. Also, if* $\hat{w}^{\mathrm{R}}\, \underset{\leftarrow}{q} \Longrightarrow_{\Phi'} \Omega'$*, then* $q \xrightarrow{w} q'$ *for some state* q' *such that* $\Omega' = \underset{\leftarrow}{q'}$*.*

Once again, this gives the adequacy of the object-oriented choreography of NFAs nearly for free.

However, there is one slight blemish to the soundness clauses. Its premise, "If $\hat{a}\,\underset{\leftarrow}{q} \longrightarrow_{\Phi'} \Omega'$ […]" deals only with ideas from the choreography and ordered rewriting. Its conclusion, however, entangles ideas from the mathematical model of the NFA ("then $q \xrightarrow{a} q'_a$ for some state q'_a […]") with ideas from the choreography ("[…] such that $\Omega' = \underset{\leftarrow}{q'_a}$."). In a way, the soundness clause violates a kind of meta-level stratification. Soundness should relate one model, on the one hand, to another model, on the other hand, but that is not what happens here.

Fortunately, it is not difficult to rephrase the soundness clause in a way that adheres to the desired meta-level stratification. Upon examining the definition of $\hat{a}$, a rewriting $\hat{a}\,\underset{\leftarrow}{q} \longrightarrow_{\Phi'} \Omega'$ exists if and, most importantly, only if $\Omega' = \underset{\leftarrow}{q'}$ for some state q'. The statement of soundness is thus equivalent to:

- If $\hat{a}\,\underset{\leftarrow}{q} \longrightarrow_{\Phi'} \underset{\leftarrow}{q'}$, then $q \xrightarrow{a} q'_a$ for some state q'_a such that $\underset{\leftarrow}{q'} = \underset{\leftarrow}{q'_a}$.

But because there is a unique atom for each state, equality of state atoms coincides with equality of the states themselves. Therefore, we can simplify the statement further and combine it with completeness:

Corollary 6.21 (Adequacy of the object-oriented NFA choreography). *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ, with choreography (θ', Φ') as described above. The following hold.*

- *$q \xrightarrow{a} q'$ if, and only if, $\hat{a}\,\underset{\leftarrow}{q} \longrightarrow_{\Phi'} \underset{\leftarrow}{q'}$.*
- *$q \in F$ if, and only if, $\hat{\$}\,\underset{\leftarrow}{q} \longrightarrow_{\Phi'} (\cdot)$.*
- *$q \xrightarrow{w} q'$ if, and only if, $\hat{w}^{\mathrm{R}}\,\underset{\leftarrow}{q} \Longrightarrow_{\Phi'} \underset{\leftarrow}{q'}$.*

6.5.3 *Incorporating NFA bisimilarity*

Recall from corollary 6.18 the nearly free adequacy result for the functional choreography of NFAs.

Corollary 6.18 (Adequacy of the functional NFA choreography). *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ, with choreography (θ, Φ) as described above. The following hold.*

- *$q \xrightarrow{a} q'_a$ only if $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_{\Phi} \hat{q}'_a$. Also, if $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_{\Phi} \Omega'$, then $q \xrightarrow{a} q'_a$ for some state q'_a such that $\Omega' = \hat{q}'_a$.*
- *$q \in F$ if, and only if, $\underset{\rightarrow}{\$}\,\hat{q} \longrightarrow_{\Phi} (\cdot)$.*
- *$q \xrightarrow{w} q'$ only if $\underset{\rightarrow}{w^{\mathrm{R}}}\,\hat{q} \Longrightarrow_{\Phi} \hat{q}'$. Also, if $\underset{\rightarrow}{w^{\mathrm{R}}}\,\hat{q} \Longrightarrow_{\Phi} \Omega'$, then $q \xrightarrow{w} q'$ for some state q' such that $\Omega' = \hat{q}'$.*

The soundness clause has the same blemish as the object-oriented choreography's soundness clause had: its conclusion violates a kind of meta-level stratification, mixing ideas from the mathematical model ("then $q \xrightarrow{a} q'_a$ for some state q'_a […]") with ideas from the choreography ("[…] such that

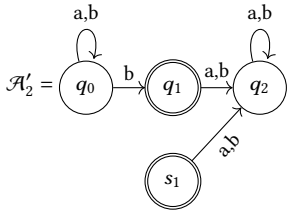
$\Omega' = \hat{q}'_a$.”). It would be much nicer if we could rephrase soundness in a stratified way.

As for the object-oriented choreography, we can begin by noticing that a rewriting $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_\Phi \Omega'$ exists if, and only if, $\Omega' = \hat{q}'$ for some state q'. The statement of soundness is thus equivalent to:

- If $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_\Phi \hat{q}'$, then $q \xrightarrow{a} q'_a$ for some state q'_a such that $\hat{q}' = \hat{q}'_a$.

This is still not quite satisfactory because the conclusion brings in choreographic ideas, namely $\hat{q}' = \hat{q}'_a$. Is it possible to characterize this relationship between q' and q'_a natively in terms of the NFA's mathematical model?

Unfortunately, this is not nearly as easy as it was for the object-oriented choreography. Unlike there, equality of state encodings does not coincide with equality of the states themselves, *i.e.*, $\hat{q}' = \hat{q}'_a$ does *not* imply $q' = q'_a$. The equirecursive treatment of coinductively defined propositions leads to a quite generous notion of equality on propositions, which in turn makes equality of state encodings a coarser equivalence than equality of the states themselves. As a concrete counterexample, consider the NFA and encoding shown in the adjacent figure; it is the same NFA as shown in fig. 6.6, but with one added state, s_1, that is unreachable from the others. In this counterexample, as a coinductive consequence of the equirecursive treatment of definitions, $\hat{q}_1 = \hat{s}_1$ but $q_1 \neq s_1$.



$$\Phi = \left(\begin{array}{l}\hat{q}_0 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_0) \mathbin{\&} (\underset{\rightarrow}{b} \setminus (\uparrow\downarrow\hat{q}_0 \mathbin{\&} \uparrow\downarrow\hat{q}_1)) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{n})\,,\\ \hat{q}_1 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{b} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{y})\,,\\ \hat{q}_2 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{b} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{n})\,,\\ \hat{s}_1 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{b} \setminus \uparrow\downarrow\hat{q}_2) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{y})\end{array}\right)$$

Figure 6.8: A slightly modified version of the NFA from fig. 6.6; and a choreography

One possible remedy for this lack of injectivity might be to revise the encoding to have a stronger nominal character. By tagging each state's encoding with an atom that is unique to that state, we can make the encoding manifestly injective. For instance, given the pairwise distinct atoms $\{\underset{\rightarrow}{q} \mid q \in F\}$ and $\{\underset{\rightarrow}{\bar{q}} \mid q \in Q - F\}$ to tag final and non-final states, respectively, we could define an alternative encoding, $\check{q}$:

$$\check{q} \triangleq (\underset{\rightarrow}{\$} \setminus \uparrow\check{\underset{\rightarrow}{F}}(q)) \mathbin{\&} \bigamp_{a\in\Sigma} \left(\underset{\rightarrow}{a} \setminus \&_{q'_a \in \Delta(q,a)} \uparrow\downarrow\check{q}'_a\right) \text{ where } \check{\underset{\rightarrow}{F}}(q) = \begin{cases} \underset{\rightarrow}{q} & \text{if } q \in F \\ \underset{\rightarrow}{\bar{q}} & \text{if } q \notin F\,. \end{cases}$$

Under this alternative encoding, the states q_1 and s_1 of fig. 6.8 are no longer a counterexample to injectivity: Because q_1 and s_1 are distinct states, they correspond to distinct tags, and so $\check{q}_1 \neq \check{s}_1$.

Although such a solution is certainly possible, it seems unsatisfyingly ad hoc. A closer examination of the preceding counterexample reveals that the states q_1 and s_1, while not equal, are in fact bisimilar (section 2.3.1). In other words, although the choreographing of states is not, strictly speaking, injective, it is injective *up to bisimilarity*: $\hat{q} = \hat{s}$ implies $q \sim s$.

THEOREM 6.22. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ. For all states q and s, if $\hat{q} = \hat{s}$, then $q \sim s$.*

Proof. We will show that the relation $\mathcal{R} = \{(q, s) \mid \hat{q} = \hat{s}\}$ is a bisimulation and is therefore included in NFA bisimilarity.

Input bisimilarity We must show that, for all input symbols $a \in \Sigma$, all $\mathcal{R}$-related states have $\mathcal{R}$-related a-successors.

Let q and s be $\mathcal{R}$-related states, which, being $\mathcal{R}$-related, have equal encodings, *i.e.*, $\hat{q} = \hat{s}$. Because definitions are treated equirecursively, their unrollings are also equal. For each state q'_a that a-succeeds q, there must therefore exist a state s'_a such that $\hat{q}'_a = \hat{s}'_a$. In other words, each a-successor of state q is $\mathcal{R}$-related to an a-successor of state s.

Finality We must show that all $\mathcal{R}$-related states have matching finalities, *i.e.*, that $q \mathcal{R} s$ implies $q \in F$ if, and only if, $s \in F$.

Let q and s be $\mathcal{R}$-related states, with q a final state. Being $\mathcal{R}$-related, the states q and s have equal encodings, *i.e.*, $\hat{q} = \hat{s}$. Because definitions are treated equirecursively, their unrollings are also equal. It follows that $\underset{\rightarrow}{F}(q) = \underset{\rightarrow}{F}(s)$, and so s is also a final state. □

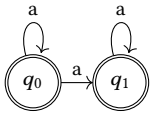
Unfortunately, the converse is not true: bisimilar NFA states do not, in general, have equal encodings. As a concrete counterexample, consider the NFA and choreography depicted in the adjacent figure. It is straightforward to verify that q_0 and q_1 are bisimilar states. But their encodings are not equal. Unrolling definitions, we have

$$\hat{q}_0 = (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_0 \mathbin{\&} \uparrow\downarrow\hat{q}_1) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\mathbf{1}) \neq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_1) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\mathbf{1}) = \hat{q}_1 \,.$$

These propositions are not equal because the former has a first clause with shape $(\underset{\rightarrow}{a} \setminus \uparrow\downarrow- \mathbin{\&} \uparrow\downarrow-)$, whereas the latter's first clause has shape $(\underset{\rightarrow}{a} \setminus \uparrow\downarrow-)$. In other words, the encodings of states q_0 and q_1 are not equal precisely because the two states have different numbers of a-successors.

In some sense, the generous equality induced by the equirecursive treatment of definitions has outpaced the remaining aspects of propositional equality. Because equality of state encodings does not coincide with bisimilarity of NFA states, we cannot easily complete our program of simplifying the statement of NFA soundness.

However, all is not lost. First, although for NFAs bisimilar states do not always have equal encodings, for *deterministic* finite automata bisimilar states do indeed have equal encodings.



$\hat{q}_0 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_0 \mathbin{\&} \uparrow\downarrow\hat{q}_1) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{y})$

$\hat{q}_1 \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{q}_1) \mathbin{\&} (\underset{\rightarrow}{\$} \setminus \uparrow\underset{\rightarrow}{y})$

Figure 6.9: An NFA that accepts all finite words over the alphabet $\Sigma = \{a\}$

Theorem 6.23. *Let $\mathcal{A} = (Q, \delta, F)$ be a* DFA *over an input alphabet Σ. For all states q and s, if $q \sim s$, then $\hat{q} = \hat{s}$.*

Proof. Because $\mathcal{A}$ is deterministic, the states q and s have unique a-successors for each input symbol $a \in \Sigma$. Because q and s are bisimilar, so are their a-successors. By the coinductive hypothesis, the unique a-successors of q and s have equal encodings: $\hat{q}'_a = \hat{s}'_a$ □

Thus, for DFAs only, we have $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_\Phi \hat{q}'$ if, and only if, $q \xrightarrow{a} \sim q'$, which is properly stratified.

Second, in the following chapter, we will develop a richer notion of propositional equivalence, *ordered rewriting bisimilarity*. In applying it to the functional choreography of NFAs (section 6.5.1), we will see that NFA states that are NFA-bisimilar have state encodings that, while not necessarily equal, are always rewriting-bisimilar; the converse will also hold. This will allow us to rephrase soundness (and completeness) of the NFA choreography to a form that is properly stratified: $\underline{a}\,\hat{q} \longrightarrow_\Phi \cong \hat{q}'$ if, and only if, $q \xrightarrow{a} \sim q'$.

7
Bisimilarity for ordered rewriting

With the shift from the global, state-transformation view of ordered rewriting put forth in chapter 5 to the local, formula-as-process view developed in the preceding chapter, we are now in a position to examine how individual propositions and, more generally, contexts behave and interact. And, in line with other, process calculi descriptions of concurrency, it is then natural to ask when two contexts have the same behavior.

We will consider two contexts to have the same behavior only if there is no experiment by an external observer that will eventually yield an observable distinction. But then what is observable? According to the formula-as-process view, each proposition has its own local thread of control. This locality, together with the output transitions defined in section 6.2, suggests what may be observed: A proposition's structure is opaque; only outward-directed *atomic* propositions that are located at the edge of an ordered context are observable.

Intuitively, for example, the contexts $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$ and $\underrightarrow{b}$ ought to be considered behaviorally equivalent: there is no circumstance under which anything other than $\underrightarrow{b}$ can be observed from $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$. In particular, the atom $\underrightarrow{a}$ at the left edge of the context $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$ cannot be observed because it is directed inward and therefore acting as an input to, not an output from, $(\underrightarrow{a} \setminus \underrightarrow{b})$.

As another example, $\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b})$ and $(\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b}$ ought to be considered behaviorally equivalent, intuitively because they are logically equivalent[1].

[1] That is, $\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b}) \dashv\vdash (\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b}$.

Following the vast literature on various forms of bisimilarity, this chapter therefore develops a notion of *ordered rewriting bisimilarity* in which atoms are observable. Section 7.1 begins by defining rewriting bisimilarity (along with a few auxiliary notions) and then presents a few examples of contexts that are *not* bisimilar under this definition.

However, it will prove to be cumbersome to use rewriting bisimilarity's definition alone to establish that two contexts are bisimilar, a problem that is familiar from process calculi bisimilarities. Therefore, in section 7.1.1, we present *labeled bisimilarity*, a sound proof technique for rewriting bisimilarity – ordered contexts that are labeled bisimilar will also be rewriting bisimilar.

Unlike the π-calculus's labeled bisimilarity, our labeled bisimilarity is, surprisingly, complete for rewriting bisimilarity – contexts that are rewriting bisimilar will also be labeled bisimilar.

This chapter concludes with several applications of ordered rewriting bisimilarity to our now-familiar running examples, binary counters and nondeterministic finite automata. Section 7.2 proves that NFA-bisimilar states have rewriting-bisimilar encodings under the functional choreography described in section 6.5.1, which allows us to finally rephrase the choreography's adequacy in a clean and stratified form. And section 7.3 proves that binary counters under the object-oriented choreography of section 6.4.1 are rewriting-bisimilar if, and only if, they have the same denotation.

7.1 *Ordered rewriting bisimilarity*

As mentioned above, we will take atoms to be observable, consistent with the local interaction semantics and output transitions defined in section 6.2. It is worth reiterating that an atom's direction and location within a larger context are crucial to its observability – only outward-directed atoms located at the edges of a context are immediately observable. For example, in $\underset{\leftarrow}{a}\,\Omega$ and $\Omega\,\underset{\rightarrow}{b}$, the atoms $\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{b}$, respectively, are observable because they are in position to be received by an observer. The observer $(\uparrow A^+ \,/\, \underset{\leftarrow}{a})$ can receive $\underset{\leftarrow}{a}$ from $\underset{\leftarrow}{a}\,\Omega$, and the observer $(\underset{\rightarrow}{b} \setminus \uparrow B^+)$ can receive $\underset{\rightarrow}{b}$ from $\Omega\,\underset{\rightarrow}{b}$:

$$(\uparrow A^+ \,/\, \underset{\leftarrow}{a})\;(\underset{\leftarrow}{a}\,\Omega) \longrightarrow A^+\,\Omega \quad \text{and} \quad (\Omega\,\underset{\rightarrow}{b})\;(\underset{\rightarrow}{b} \setminus \uparrow B^+) \longrightarrow \Omega\,B^+\,.$$

But these same atoms cannot be observed in $\Omega\,\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{b}\,\Omega$ when Ω is nonempty, precisely because their new locations do not complement their directions. Given the syntactic restriction imposed on left- and right-hand implications, no implication placed to the right of $\underset{\leftarrow}{a}$ will be able to consume $\underset{\leftarrow}{a}$; likewise for implications placed to the left of $\underset{\rightarrow}{b}$. In other words, the observable atoms are exactly those atoms that are immediately outgoing.

Ordered rewriting is asynchronous: $(\uparrow A^+ \,/\, \underset{\leftarrow}{a})\;(\underset{\leftarrow}{a} \bullet B^+) \Longrightarrow A^+\,B^+$ is not possible in a single, synchronous step. We should therefore expect our notion of rewriting bisimilarity to have some analogy to the kinds of bisimilarities developed for the asynchronous π-calculus[2] and asynchronous CCS.[3] In particular, that means that input processes or, in our case, implications ought not to be directly observable.

Instead, the asynchronous nature of rewriting leads us to the type of experiment that external observers may perform: provide two contexts with some incoming messages (or none at all) and observe what outgoing messages, if any, are eventually produced. If the two contexts eventually produce different outgoing messages, then those contexts are observably distinguishable and cannot be considered behaviorally equivalent.

Lastly, we will not consider the time or number of computational steps to be observable[4] – all that matters is whether, given the same inputs, the same

[2] Amadio, Castellani, et al. 1998.

[3] Boreale et al. 2002.

[4] Nor do we consider divergence to be observable, so we do not pursue testing equivalence, only bisimilarity.

outputs are eventually produced. The resulting bisimilarity will therefore be a weak bisimilarity.

So, to summarize, our ordered rewriting bisimilarity will be an asynchronous, weak bisimilarity with two conditions: output bisimulation and input bisimulation.[5] This combination of asynchronous weak bisimilarity with observable atoms means that our definition of ordered rewriting bisimilarity will be similar to Deng et al.'s contextual preorder for linear logic.[6] Besides the obvious difference in structural properties (ordered and linear) and type of relation (bisimulation and simulation; equivalence and preorder), the particulars of our definition will be different.

[5] We could combine the two conditions into a monolithic one, but that becomes rather unwieldy.

[6] Deng et al. 2016.

We are nearly ready to define ordered rewriting bisimilarity, but we first must define a few auxiliary relations on contexts.

Definition 7.1 (Framed binary relations). Let $\mathcal{R}$ be a binary relation on ordered contexts. Given ordered contexts Δ_L and Δ_R, let $(\Delta_L \mathcal{R} \Delta_R)$ be the least binary relation such that:

$$\frac{(\Omega = \Delta_L\ \Omega_0\ \Delta_R) \quad \Omega_0\ \mathcal{R}\ \Omega_0' \quad (\Delta_L\ \Omega_0'\ \Delta_R = \Omega')}{\Omega\ (\Delta_L \mathcal{R} \Delta_R)\ \Omega'}$$

In other words, $(\Delta_L \mathcal{R} \Delta_R)$ relates contexts consisting of $\mathcal{R}$-related middles that are each surrounded by Δ_L and Δ_R.

Furthermore, let $[\mathcal{R}]$ be the input contextual closure of $\mathcal{R}$ – that is, $[\mathcal{R}]$ is the least binary relation such that:

$$\frac{\Omega\ \mathcal{R}\ \Delta}{\Omega\ [\mathcal{R}]\ \Delta} \qquad \frac{\Omega\ [\mathcal{R}]\ \Delta}{\underrightarrow{a}\,\Omega\ [\mathcal{R}]\ \underrightarrow{a}\,\Delta} \qquad \frac{\Omega\ [\mathcal{R}]\ \Delta}{\Omega\,\underleftarrow{a}\ [\mathcal{R}]\ \Delta\,\underleftarrow{a}}$$

Equivalently, $\Omega\ [\mathcal{R}]\ \Delta$ if, and only if, there exist input contexts $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$ such that $\Omega\ (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R)\ \Delta$.

With these auxiliary relations in hand, we can now turn to defining ordered rewriting bisimilarity. We will state its definition first and then justify that definition on the basis of indistinguishability of observations.

Definition 7.2. A *rewriting bisimulation*, $\mathcal{R}$, is a symmetric binary relation among contexts that satisfies the following conditions.

Output bisimulation If $\Omega\ \mathcal{R} \Longrightarrow \underleftarrow{\Delta}_L'\ \Delta'\ \underleftarrow{\Delta}_R'$, then $\Omega \Longrightarrow (\underleftarrow{\Delta}_L' \mathcal{R} \underleftarrow{\Delta}_R')\ \underleftarrow{\Delta}_L'\ \Delta'\ \underleftarrow{\Delta}_R'$.

Input bisimulation If $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R\ (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R) \Longrightarrow \Delta'$, then $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R \Longrightarrow \mathcal{R}\ \Delta'$.

Then *rewriting bisimilarity*, $\cong$, is the largest rewriting bisimulation.

Notice that a third, reduction bisimulation property is a trivial instance of the output and input bisimulation conditions – namely when the output and input contexts, $\underleftarrow{\Delta}_L'$ and $\underleftarrow{\Delta}_R'$ and $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$, respectively, are empty:

Theorem 7.1. *If $\mathcal{R}$ is a rewriting bisimulation, then $\mathcal{R}$ satisfies*

Reduction bisimulation *If $\Omega\ \mathcal{R} \Longrightarrow \Delta'$, then $\Omega \Longrightarrow \mathcal{R}\ \Delta'$.*

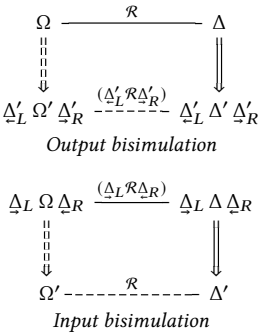


Figure 7.1: Rewriting bisimulation conditions, in diagrams

The clauses of definition 7.2 could do with some explanation. Let's begin with the output bisimulation condition.

Expanding slightly, we are given that there exists a context Δ such that $\Omega \mathrel{\mathcal{R}} \Delta \Longrightarrow \underleftarrow{\Delta}'_L \, \Delta' \, \underrightarrow{\Delta}'_R$. Based on the local interaction semantics (section 6.2), this means *(i)* that Δ can eventually output $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$ and then continue as Δ'; and *(ii)* that Ω is $\mathcal{R}$-related to Δ. For $\mathcal{R}$ to be a rewriting bisimulation, the context Ω ought to be able to simulate Δ's eventual output of $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$, otherwise the $\mathcal{R}$-related contexts Ω and Δ could be distinguished based on their (eventual) output behavior. Moreover, the continuations ought to be $\mathcal{R}$-related as well. Formally, we ought to have $\Omega \Longrightarrow \underleftarrow{\Delta}'_L \, \Omega' \, \underrightarrow{\Delta}'_R$ (*i.e.*, Ω eventually outputs $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$) and $\Omega' \mathrel{\mathcal{R}} \Delta'$, for some context Ω', which is all neatly packaged up as $\Omega \Longrightarrow (\underleftarrow{\Delta}'_L \mathcal{R} \underrightarrow{\Delta}'_R) \; \underleftarrow{\Delta}'_L \, \Delta' \, \underrightarrow{\Delta}'_R$.

Input bisimulation is dual to output bisimulation. For input bisimulation, we are given that $\underrightarrow{\Delta}_L \, \Omega \, \underleftarrow{\Delta}_R \; (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R) \Longrightarrow \Delta'$. Expanding slightly, there exists a context Δ such that $\Omega \mathrel{\mathcal{R}} \Delta$ and $\underrightarrow{\Delta}_L \, \Delta \, \underleftarrow{\Delta}_R \Longrightarrow \Delta'$. In other words, once provided with the incoming messages $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$, the context Δ can eventually evolve to Δ'. Being $\mathcal{R}$-related to Δ, the context Ω, when provided with the same incoming messages, must be able to evolve to a context that is $\mathcal{R}$-related to Δ', otherwise Ω and Δ could be distinguished by how they react to $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$. That is, we must have $\underrightarrow{\Delta}_L \, \Omega \, \underleftarrow{\Delta}_R \Longrightarrow \Omega' \mathrel{\mathcal{R}} \Delta'$ for some Ω', which is neatly packaged as $\underrightarrow{\Delta}_L \, \Omega \, \underleftarrow{\Delta}_R \Longrightarrow \mathcal{R} \, \Delta'$.

The other way to understand rewriting bisimilarity is by analogy with the asynchronous CCS's notion of weak bisimilarity.[7] There, a weak bisimulation can be described as a symmetric relation $\mathcal{R}$ on processes that satisfies three conditions:[8]

- If $P \mathrel{\mathcal{R}} \overset{\bar{c}}{\Longrightarrow} Q'$, then $P \overset{\bar{c}}{\Longrightarrow} \mathcal{R} \, Q'$.
- If $P \mathrel{\mathcal{R}} \overset{\tau}{\Longrightarrow} Q'$, then $P \overset{\tau}{\Longrightarrow} \mathcal{R} \, Q'$.
- If $P \mathrel{\mathcal{R}} \overset{c}{\Longrightarrow} Q'$, then either $P \overset{c}{\Longrightarrow} \mathcal{R} \, Q'$ or there exists a process P' such that $P \overset{\tau}{\Longrightarrow} P'$ and $\bar{c} \mid P' \mathrel{\mathcal{R}} Q'$.

Weak bisimilarity for the asynchronous CCS is then the largest such bisimulation.

The first of these three conditions corresponds to rewriting bisimilarity's output bisimulation condition with nonempty output contexts $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$. As mentioned in section 6.2, $\Delta \Longrightarrow \underleftarrow{\Delta}'_L \, \Delta' \, \underrightarrow{\Delta}'_R$ functions as an implicit weak output transition from Δ to Δ', with outputs $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$. Similarly, $\Omega \Longrightarrow (\underleftarrow{\Delta}'_L \mathcal{R} \underrightarrow{\Delta}'_R) \; \underleftarrow{\Delta}'_L \, \Delta' \, \underrightarrow{\Delta}'_R$ is analogous to $P \overset{\bar{c}}{\Longrightarrow} \mathcal{R} \, Q'$.

The second of the three conditions imposed by asynchronous CCS weak bisimilarity corresponds to our rewriting bisimilarity's reduction bisimulation property (theorem 7.1), which is really just either output or input bisimulation with empty output contexts $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$ or input contexts $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$.

The third of the three conditions imposed by asynchronous CCS weak bisimilarity corresponds to rewriting bisimilarity's input bisimulation condition.

[7] Amadio, Castellani, et al. 1998; Boreale et al. 2002.

[8] The premises of these conditions are usually stated with strong transitions, but we prefer this phrasing for its similarity to rewriting bisimilarity.

The CCS condition is equivalent to "If $P \ \mathcal{R} \xrightarrow{c} Q'$, then $\bar{c} \mid P \overset{\tau}{\Longrightarrow} \mathcal{R} \ Q'$." (In fact, it is typical to use this phrasing in the definition of asynchronous CCS bisimilarity.) Because asynchronous CCS weak bisimilarity is a congruence, that condition can be rephrased as "If $P \ \mathcal{R} \ Q$ and $\bar{c} \mid Q \xrightarrow{\tau} Q'$, then $\bar{c} \mid P \overset{\tau}{\Longrightarrow} \mathcal{R} \ Q'$" without affecting the resulting bisimilarity. And, in that form, the correspondence with rewriting bisimilarity's input bisimulation condition becomes apparent.

Rewriting bisimilarity imposes very strong conditions upon bisimilar contexts, quantifying over all traces and all output and input contexts. Combined with the coinductive nature of bisimilarity, this results in a rather fine-grained equivalence. Some contexts that might, at first glance, seem like they ought to be equivalent are, in fact, not bisimilar.

- The contexts $\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a}$ and $(\cdot)$ are *not* bisimilar. Suppose, for the sake of contradiction, that they are bisimilar. Framing $\underset{\leftarrow}{b}$ onto the right, we have $(\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a})\ \underset{\leftarrow}{b} \ (\cong \underset{\leftarrow}{b})\ \underset{\leftarrow}{b}$. Composing the input and output bisimulation conditions, $(\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a})\ \underset{\leftarrow}{b} \Longrightarrow (\underset{\leftarrow}{b} \cong)\ \underset{\leftarrow}{b}$ must follow. However, this is impossible: $(\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a})\ \underset{\leftarrow}{b}$ is irreducible and does not expose $\underset{\leftarrow}{b}$ at its left end. Therefore, $\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a}$ and $(\cdot)$ *cannot* be bisimilar.

- The contexts $\underset{\rightarrow}{a}$ and $\underset{\rightarrow}{a} \,\&\, \underset{\rightarrow}{b}$ are not bisimilar. The context $\underset{\rightarrow}{a} \,\&\, \underset{\rightarrow}{b}$ can output $\underset{\rightarrow}{b}$ at its right end: $\underset{\rightarrow}{a} \,\&\, \underset{\rightarrow}{b} \longrightarrow \underset{\rightarrow}{b}$. But $\underset{\rightarrow}{a}$ cannot simulate that output – the output bisimulation condition demands $\underset{\rightarrow}{a} \Longrightarrow (\cong \underset{\rightarrow}{b})\ \underset{\rightarrow}{b}$, which is impossible.

- If we were working in an unfocused framework, the contexts $\underset{\rightarrow}{a}$ and $\underset{\rightarrow}{a} \,\&\, \top$ would not be bisimilar. The context $\underset{\rightarrow}{a} \,\&\, \top$ would reduce (*i.e.*, $\underset{\rightarrow}{a} \,\&\, \top \longrightarrow \top$), and so the input bisimulation condition and the irreducibility of $\underset{\rightarrow}{a}$ would imply $\underset{\rightarrow}{a} \cong \top$. The output bisimulation condition would then demand that $\top$ expose $\underset{\leftarrow}{a}$ at its left end, which is impossible. As we will see later, $\underset{\rightarrow}{a}$ and $\underset{\rightarrow}{a} \,\&\, \top$ are bisimilar in a focused framework.

Now we would like to confirm our earlier intuition about the equivalence of $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \,/\, \underset{\rightarrow}{b})$ and $\underset{\rightarrow}{b}$ by proving that $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \,/\, \underset{\rightarrow}{b}) \cong \underset{\rightarrow}{b}$. Unfortunately, the definition of rewriting bisimilarity is not immediately suitable for establishing that two contexts are bisimilar. The output and input bisimulation conditions are so strong that they become difficult to prove directly. For instance, to establish $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \cong \underset{\rightarrow}{b}$, we would need to prove that:

Input bisimulation $\underset{\rightarrow}{\Delta}_L\ \underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})\ \underset{\leftarrow}{\Delta}_R \Longrightarrow \Delta'$ implies $\underset{\rightarrow}{\Delta}_L\ \underset{\rightarrow}{b}\ \underset{\leftarrow}{\Delta}_R \cong \Delta'$; and symmetrically, $\underset{\rightarrow}{\Delta}_L\ \underset{\rightarrow}{b}\ \underset{\leftarrow}{\Delta}_R \Longrightarrow \Delta'$ implies $\underset{\rightarrow}{\Delta}_L\ \underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})\ \underset{\leftarrow}{\Delta}_R \cong \Delta'$;

Output bisimulation $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \Longrightarrow \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$ implies $\underset{\rightarrow}{b} \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \cong \underset{\rightarrow}{\Delta}'_R)\ \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$; and symmetrically, $\underset{\rightarrow}{b} \Longrightarrow \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$ implies $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \cong \underset{\rightarrow}{\Delta}'_R)\ \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$.

In this small example, it is possible to imagine tediously proving these statements – after all, there are not that many traces involving $\underset{\rightarrow}{a}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$. However, in general, a proof technique for rewriting bisimilarity is sorely needed.

7.1.1 *A labeled proof technique for rewriting bisimilarity*

In CCS and the π-calculus, bisimilarity is similarly too strong to be used directly in proving the equivalence of processes. There, a sound proof technique for bisimilarity is built around a labeled transition system and a notion of labeled bisimulation.[9] Because the labeled transition system is image-finite, proving that two processes are labeled bisimilar is more tractable than directly proving them to be bisimilar.

[9] Sangiorgi and Walker 2003.

In this section, we follow that strategy and develop *labeled bisimilarity* as a sound – and, surprisingly, also complete – proof technique for rewriting bisimilarity. Like its CCS and π-calculus analogues, labeled bisimilarity is more tractable than rewriting bisimilarity because it uses individual input transitions in place of full rewriting sequences.

Instead of defining labeled bisimulations directly, we use a refactorization, standard in the study of up-to techniques,[10] in which we first define a notion of *progression* and then characterize labeled bisimulations in terms of progression.

[10] Pous and Sangiorgi 2011.

Definition 7.3. A binary relation $\mathcal{R}$ on contexts *progresses* to binary relation $\mathcal{S}$ if $\mathcal{R}$ is symmetric and the two relations satisfy the following conditions.

Immediate output bisim. If $\Omega\ \mathcal{R}\ \Delta = \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$, then $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{S} \underset{\rightarrow}{\Delta}'_R)\ \Delta$.

Immediate input bisimulation If $\Omega\ \mathcal{R}\ \Delta$ and $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$, then $\underset{\rightarrow}{\Delta}_L\ \Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow \mathcal{S}\ \Delta'$.

Reduction bisimulation If $\Omega\ \mathcal{R} \longrightarrow \Delta'$, then $\Omega \Longrightarrow \mathcal{S}\ \Delta'$.

Emptiness bisimulation If $\Omega\ \mathcal{R}\ (\cdot)$, then: $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow (\mathcal{S}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ for all $\underset{\rightarrow}{\Delta}$; and $\Omega\ \underset{\leftarrow}{\Delta} \Longrightarrow (\underset{\leftarrow}{\Delta}\mathcal{S})\ \underset{\leftarrow}{\Delta}$ for all $\underset{\leftarrow}{\Delta}$.

A *labeled bisimulation* is a relation that progresses to itself, and *labeled bisimilarity* is the largest labeled bisimulation.

The immediate output, immediate input, and reduction bisimulation conditions are all single-step forms of rewriting bisimilarity's output and input bisimulation conditions (definition 7.2) and reduction bisimulation property (theorem 7.1). The emptiness bisimulation condition, on the other hand, is necessary for labeled bisimilarity to be complete.[11] Emptiness bisimulation is equivalent to requiring that $\Omega\ \mathcal{R}\ (\cdot)$ implies both $\Omega \Longrightarrow (\cdot)$ and $(\cdot)\ \mathcal{S}\ (\cdot)$. (See theorem A.1 for a proof sketch.) In this way, it can be seen that being $\mathcal{R}$-related to $(\cdot)$ is possible only if Ω is morally equivalent to $(\cdot)$, in the sense that Ω must be able to spontaneously evolve to $(\cdot)$.

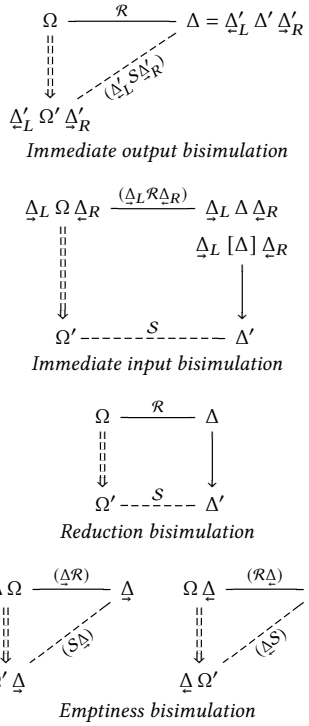


Figure 7.2: Labeled bisimulation conditions, in diagrams

[11] A similar condition appears in the contextual preorder of Deng et al. (2016).

It is relatively straightforward to show that labeled bisimilarity is complete with respect to rewriting bisimilarity: every rewriting bisimulation is itself a labeled bisimulation.

Theorem 7.2 (Completeness of labeled bisimilarity). *Every rewriting bisimulation is also a labeled bisimulation, and labeled bisimilarity consequently contains rewriting bisimilarity.*

Proof. Let $\mathcal{R}$ be a rewriting bisimulation. The immediate output, immediate input, and reduction bisimulation conditions are trivial instances of the output and input bisimulation conditions. For instance, to prove that $\mathcal{R}$ is an immediate input bisimulation, assume that $\Omega\ \mathcal{R}\ \Delta$ and $\underrightarrow{\Delta}_L\,[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$; then $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R\ (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R) \longrightarrow \Delta'$. Because $\mathcal{R}$ is a rewriting bisimulation, it follows from the input bisimulation property that $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R \Longrightarrow_{\mathcal{R}} \Delta'$.

The emptiness bisimulation condition follows from the composition of the input bisimulation property with the output bisimulation property. □

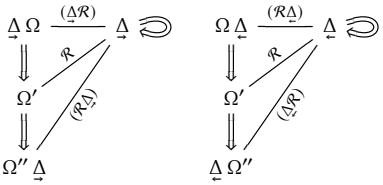


Figure 7.3: Emptiness bisimulation property as a consequence of input and output bisimulation properties

Unfortunately, the direct converse is not true: a labeled bisimulation is not necessarily itself a rewriting bisimulation. For example, consider the least symmetric binary relation $\mathcal{R}$ such that

$$\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b})\ \mathcal{R}\ (\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b} \quad \text{and} \quad \underrightarrow{c}\ \mathcal{R}\ \underrightarrow{c} \quad \text{and} \quad (\cdot)\ \mathcal{R}\ (\cdot)\,.$$

The relation $\mathcal{R}$ is a labeled bisimulation, but it does not qualify as a rewriting bisimulation because it does not satisfy the more general input bisimulation condition: for instance, $\underrightarrow{a}\,(\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b}))\ (\underrightarrow{a}\mathcal{R})\ \underrightarrow{a}\,((\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b})$ does not imply $\underrightarrow{a}\,(\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b})) \Longrightarrow_{\mathcal{R}} \underrightarrow{a}\,((\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b})$. That would be possible only if $\underrightarrow{a}\,(\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b}))$ and $\underrightarrow{a}\,((\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b})$ were $\mathcal{R}$-related.

Even though a labeled bisimulation itself is not a rewriting bisimulation, a slightly weaker statement is nevertheless true: each labeled bisimulation is contained within *some* rewriting bisimulation. Specifically, if $\mathcal{R}$ is a labeled bisimulation, then its input contextual closure, $[\mathcal{R}]$, as described in definition 7.1 is such a rewriting bisimulation. Fortunately, this will be enough to prove that labeled bisimilarity is sound.

The proof of soundness is clean, but it does wind through a few lemmas. The first of these describes a condition under which the union of two relations is a labeled bisimulation.

Lemma 7.3. *Let $\mathcal{S}$ be a labeled bisimulation. If $\mathcal{R}$ progresses to $\mathcal{R} \cup \mathcal{S}$, then $\mathcal{R} \cup \mathcal{S}$ is also a labeled bisimulation.*

Proof. When $\mathcal{S}$ is a labeled bisimulation and $\mathcal{R}$ progresses to $\mathcal{R} \cup \mathcal{S}$, then the relation $\mathcal{R} \cup \mathcal{S}$ progresses to itself, *i.e.*, $\mathcal{R} \cup \mathcal{S}$ is a labeled bisimulation. If $\Omega\ (\mathcal{R} \cup \mathcal{S})\ \Delta$ because Ω and Δ are $\mathcal{R}$-related, then the conditions for progressing to $\mathcal{R} \cup \mathcal{S}$ are satisfied by $\mathcal{R}$ progressing to $\mathcal{R} \cup \mathcal{S}$. If, on the other hand, $\Omega\ (\mathcal{R} \cup \mathcal{S})\ \Delta$ because Ω and Δ are $\mathcal{S}$-related, then the conditions for progressing to $\mathcal{R} \cup \mathcal{S}$ are satisfied by the fact that $\mathcal{S}$ is a labeled bisimulation. □

Next, we use this result to prove that framing a single input atom onto a labeled bisimulation results in a binary relation that does not stray too far from a labeled bisimulation.

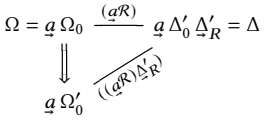
Lemma 7.4. *If $\mathcal{R}$ is a labeled bisimulation, then so are $(\underrightarrow{a}\mathcal{R}) \cup \mathcal{R}$ and $(\mathcal{R}\underleftarrow{a}) \cup \mathcal{R}$, for all $\underrightarrow{a}$ and $\underleftarrow{a}$, respectively.*

Proof. Let $\mathcal{R}$ be a labeled bisimulation. We shall prove that $(\underrightarrow{a}\mathcal{R}) \cup \mathcal{R}$ is a labeled bisimulation; the proof for $(\mathcal{R}\underleftarrow{a}) \cup \mathcal{R}$ is symmetric.

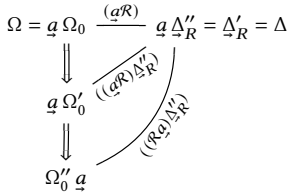
According to lemma 7.3, because $\mathcal{R}$ is a labeled bisimulation, it suffices to show that $(\underrightarrow{a}\mathcal{R})$ progresses to $(\underrightarrow{a}\mathcal{R}) \cup \mathcal{R}$. We prove each property in turn.

Immediate output bisimulation Assume that $\Omega\ (\underrightarrow{a}\mathcal{R})\ \Delta = \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$; we must show that $\Omega \Longrightarrow\big(\underleftarrow{\Delta}'_L((\underrightarrow{a}\mathcal{R}) \cup \mathcal{R})\underrightarrow{\Delta}'_R\big)\ \Delta$. Because the input atom $\underrightarrow{a}$ cannot be unified with the output atoms $\underleftarrow{\Delta}'_L$, the context $\underleftarrow{\Delta}'_L$ must be empty. We distinguish cases on the size of Δ'.

- Consider the case in which Δ' is nonempty. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate output bisimulation property after framing off $\underrightarrow{a}$ and deduce that $\Omega\ \big(\underrightarrow{a}(\Longrightarrow(\mathcal{R}\underrightarrow{\Delta}'_R))\big)\ \Delta$. Reduction is closed under framing, so we conclude that $\Omega \Longrightarrow((\underrightarrow{a}\mathcal{R})\underrightarrow{\Delta}'_R)\ \Delta$, as required.

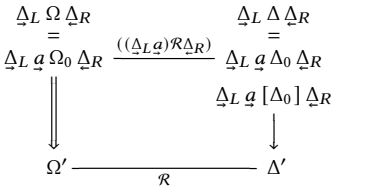


- Consider the case in which Δ' is empty – that is, the case in which $\Omega\ (\underrightarrow{a}\mathcal{R})\ \Delta = \underrightarrow{\Delta}'_R = \underrightarrow{a}\,\underrightarrow{\Delta}''_R$ for some $\underrightarrow{\Delta}''_R$. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate output bisimulation property after framing off $\underrightarrow{a}$ and deduce that $\Omega\ (\underrightarrow{a}(\Longrightarrow(\mathcal{R}\underrightarrow{\Delta}''_R)))\ \underrightarrow{\Delta}'_R$. Reduction is closed under framing, so $\Omega \Longrightarrow((\underrightarrow{a}\mathcal{R})\underrightarrow{\Delta}''_R)\ \underrightarrow{\Delta}'_R$. After framing off $\underrightarrow{\Delta}''_R$, we may subsequently appeal to the emptiness bisimulation property of $\mathcal{R}$ and deduce that $\Omega \Longrightarrow((\Longrightarrow(\mathcal{R}\underrightarrow{a}))\underrightarrow{\Delta}''_R)\ \underrightarrow{\Delta}'_R$. Once again, reduction is closed under framing, so we conclude that $\Omega \Longrightarrow(\mathcal{R}\underrightarrow{\Delta}'_R)\ \Delta$, as required.

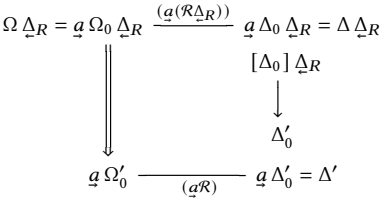


Immediate input bisimulation Assume that $\Omega\ (\underrightarrow{a}\mathcal{R})\ \Delta$ and $\underrightarrow{\Delta}_L\ [\Delta]\ \underleftarrow{\Delta}_R \longrightarrow \Delta'$; we must show that $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R \Longrightarrow((\underrightarrow{a}\mathcal{R}) \cup \mathcal{R})\ \Delta'$. According to lemma 6.5, there are two cases: either $\underrightarrow{a}$ satisfies an input demand, or it does not participate in the given input transition.

- Consider the case in which $\underrightarrow{a}$ does participate in the input transition – that is, the case in which $\Omega\ (\underrightarrow{a}\mathcal{R})\ \underrightarrow{a}\,\Delta_0 = \Delta$ and $\underrightarrow{\Delta}_L\,\underrightarrow{a}\,[\Delta_0]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$, for some Δ_0. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate input bisimulation property and deduce $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R \Longrightarrow_{\mathcal{R}} \Delta'$, as required.

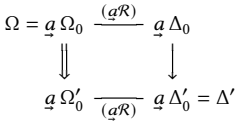


- Consider the case in which $\underrightarrow{a}$ does not participate in the input transition – that is, the case in which $\underrightarrow{\Delta}_L$ is empty and $\Omega\ (\underrightarrow{a}\mathcal{R})\ \underrightarrow{a}\,\Delta_0 = \Delta$ and $[\Delta_0]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'_0$ and $\Delta' = \underrightarrow{a}\,\Delta'_0$, for some Δ_0 and Δ'_0. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate input bisimulation property after framing off $\underrightarrow{a}$ and deduce that $\Omega\,\underleftarrow{\Delta}_R\ (\underrightarrow{a}(\Longrightarrow\mathcal{R}))\ \Delta'$. Reduction is closed under framing, so we conclude that $\Omega\,\underleftarrow{\Delta}_R \Longrightarrow(\underrightarrow{a}\mathcal{R})\ \Delta'$, as required.

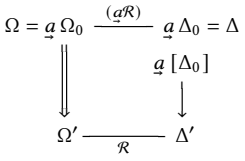


Reduction bisimulation Assume that $\Omega\ (\underrightarrow{a}\mathcal{R}) \longrightarrow \Delta'$ holds; we must show that $\Omega \Longrightarrow((\underrightarrow{a}\mathcal{R}) \cup \mathcal{R})\ \Delta'$. We distinguish cases on the origin of the given reduction.

- Consider the case in which the reduction arises from the $\mathcal{R}$-related component alone – that is, the case in which $\Omega\ (\underrightarrow{a}(\mathcal{R}\longrightarrow))\ \Delta'$. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its reduction bisimulation property after framing off $\underrightarrow{a}$ and deduce that $\Omega\ (\underrightarrow{a}(\Longrightarrow\mathcal{R}))\ \Delta'$. Reduction is closed under framing, so we conclude that $\Omega \Longrightarrow(\underrightarrow{a}\mathcal{R})\ \Delta'$, as required.



- Consider the case in which the reduction arises from an input transition on the $\mathcal{R}$-related component – that is, the case in which $\Omega\ (\underrightarrow{a}\mathcal{R})\ \underrightarrow{a}\,\Delta_0 = \Delta$ and $\underrightarrow{a}\,[\Delta_0] \longrightarrow \Delta'$, for some Δ_0. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate input bisimulation property and deduce that $\Omega \Longrightarrow\mathcal{R}\ \Delta'$, as required.



Emptiness bisimulation Assume that $\Omega\ (\underrightarrow{a}\mathcal{R})\ (\cdot)$. This is, in fact, impossible because the empty context does not contain $\underrightarrow{a}$. □

Having proved the preceding lemma about framing a single input atom, we can apply it inductively to prove that framing input contexts preserves labeled bisimulations.

LEMMA 7.5. *If $\mathcal{R}$ is a labeled bisimulation, then so is $[\mathcal{R}]$.*

Proof. Let $(\mathcal{S}_n)_{n\in\mathbb{N}}$ be the indexed family of relations given by

$$\mathcal{S}_0 = \mathcal{R}$$
$$\mathcal{S}_{n+1} = \left(\bigcup_{\underrightarrow{a}} (\underrightarrow{a}\mathcal{S}_n)\right) \cup \left(\bigcup_{\underleftarrow{a}} (\mathcal{S}_n\underleftarrow{a})\right) \cup S_n\,.$$

It is easy to prove by structural induction that each $[\mathcal{R}]$-related pair of contexts is also $\mathcal{S}_n$-related for some natural number n; and so $[\mathcal{R}]$ is contained within $\bigcup_{n=0}^{\infty}\mathcal{S}_n$. Conversely, using lemma 7.4, it is equally easy to prove by induction on n that each $\mathcal{S}_n$ is contained within $[\mathcal{R}]$ and, moreover, that each $\mathcal{S}_n$ is a labeled bisimulation.

Because each $\mathcal{S}_n$ is a labeled bisimulation, so is their least upper bound, namely $\bigcup_{n=0}^{\infty}\mathcal{S}_n = [\mathcal{R}]$. □

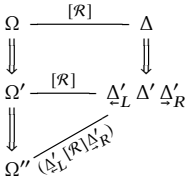
Now we use this lemma to prove that $[\mathcal{R}]$ is a rewriting bisimulation if $\mathcal{R}$ is a labeled bisimulation.

THEOREM 7.6. *If $\mathcal{R}$ is a labeled bisimulation, then rewriting bisimilarity contains $\mathcal{R}$.*

Proof. Let $\mathcal{R}$ be a labeled bisimulation. By lemma 7.5, so is $[\mathcal{R}]$. The relation $[\mathcal{R}]$ is also a rewriting bisimulation, as we will show by proving each property in turn. (Notice, too, that $[\mathcal{R}]$ is symmetric because $\mathcal{R}$ is.)

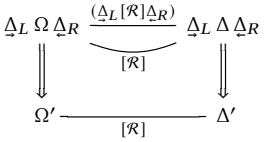
Output bisimulation Assume that $\Omega\ [\mathcal{R}]\Longrightarrow\ \underleftarrow{\Delta}'_L\,\Delta'\,\underrightarrow{\Delta}'_R$; we must show that $\Omega \Longrightarrow(\underleftarrow{\Delta}'_L[\mathcal{R}]\underrightarrow{\Delta}'_R)\ \underleftarrow{\Delta}'_L\,\Delta'\,\underrightarrow{\Delta}'_R$.

As a labeled bisimulation, $[\mathcal{R}]$ satisfies the reduction bisimulation property, so we deduce that $\Omega \Longrightarrow[\mathcal{R}]\ \underleftarrow{\Delta}'_L\,\Delta'\,\underrightarrow{\Delta}'_R$. The relation $[\mathcal{R}]$ also satisfies the immediate output bisimulation property, so we conclude that $\Omega \Longrightarrow(\underleftarrow{\Delta}'_L[\mathcal{R}]\underrightarrow{\Delta}'_R)\ \underleftarrow{\Delta}'_L\,\Delta'\,\underrightarrow{\Delta}'_R$, as required.



Input bisimulation Assume that $\underset{\rightarrow}{\Delta}_L\, \Omega\, \underset{\leftarrow}{\Delta}_R$ $(\underset{\rightarrow}{\Delta}_L[\mathcal{R}]\underset{\leftarrow}{\Delta}_R) \Longrightarrow \Delta'$; we must show that $\underset{\rightarrow}{\Delta}_L\, \Omega\, \underset{\leftarrow}{\Delta}_R \Longrightarrow [\mathcal{R}]\ \Delta'$.

Because $[\mathcal{R}]$ is input contextual, we deduce that $\underset{\rightarrow}{\Delta}_L\, \Omega\, \underset{\leftarrow}{\Delta}_R\ [\mathcal{R}] \Longrightarrow \Delta'$. As a labeled bisimulation, $[\mathcal{R}]$ satisfies the reduction bisimulation property, so we conclude that $\underset{\rightarrow}{\Delta}_L\, \Omega\, \underset{\leftarrow}{\Delta}_R \Longrightarrow [\mathcal{R}]\ \Delta'$, as required. □



Rewriting bisimilarity therefore contains every labeled bisimulation and, in particular, the largest labeled bisimulation, namely labeled bisimilarity.

Corollary 7.7. *Labeled bisimilarity is sound and complete with respect to rewriting bisimilarity.*

As a simple example of this labeled bisimilarity proof technique for rewriting bisimilarity, we shall now establish that $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$ and $\underset{\rightarrow}{b}$ are rewriting-bisimilar contexts. Let $\mathcal{R}$ be the least symmetric binary relation for which $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})\ \mathcal{R}\ \underset{\rightarrow}{b}$ and $\underset{\rightarrow}{b}\ \mathcal{R}\ \underset{\rightarrow}{b}$ and $(\cdot)\ \mathcal{R}\ (\cdot)$ hold. The relation $\mathcal{R}$ is a labeled bisimulation:

- The immediate output bisimulation condition holds because $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$ can simulate $\underset{\rightarrow}{b}$'s output of $\underset{\rightarrow}{b}$ (with $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \longrightarrow (\mathcal{R}\underset{\rightarrow}{b})\ \underset{\rightarrow}{b}$) and the former makes no immediate outputs of its own. Moreover, $\underset{\rightarrow}{b}$ and $\underset{\rightarrow}{b}$ can simulate each other's output of $\underset{\rightarrow}{b}$.
- The immediate input bisimulation condition holds vacuously for the relation $\mathcal{R}$ because neither $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$ nor $\underset{\rightarrow}{b}$ accept any inputs on either side.
- The reduction bisimulation condition holds because $\underset{\rightarrow}{b}$ can simulate the reduction $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \longrightarrow \underset{\rightarrow}{b}$ trivially (with $\underset{\rightarrow}{b} \Longrightarrow \mathcal{R}\ \underset{\rightarrow}{b}$).
- The emptiness bisimulation condition holds trivially: $\underset{\rightarrow}{\Delta} \Longrightarrow (\mathcal{R}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ for all $\underset{\rightarrow}{\Delta}$ because $(\cdot)\ \mathcal{R}\ (\cdot)$, and symmetrically for all $\underset{\leftarrow}{\Delta}$.

We may conclude from the above proof technique (theorem 7.6) that $\mathcal{R}$ is contained within rewriting bisimilarity and that $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$ and $\underset{\rightarrow}{b}$ are indeed bisimilar.

We can similarly prove that $\underset{\rightarrow}{a} \setminus (\underset{\rightarrow}{c} / \underset{\leftarrow}{b})$ and $(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b}$ are rewriting-bisimilar by showing that the least symmetric relation $\mathcal{R}$ such that $\underset{\rightarrow}{a} \setminus (\underset{\rightarrow}{c} / \underset{\leftarrow}{b})\ \mathcal{R}\ (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b}$ and $\underset{\rightarrow}{c}\ \mathcal{R}\ \underset{\rightarrow}{c}$ and $(\cdot)\ \mathcal{R}\ (\cdot)$ is a labeled bisimulation.

Somewhat surprisingly, even $\underset{\rightarrow}{a} \setminus \uparrow\downarrow(\underset{\rightarrow}{c} / \underset{\leftarrow}{b})$ and $\uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b}$ are bisimilar. This one is rather surprising because the $\uparrow\downarrow$ shift is placed in two different locations: over $- / \underset{\leftarrow}{b}$ in the former, and over $\underset{\rightarrow}{a} \setminus -$ in the latter. One might expect that the placement of $\uparrow\downarrow$ and the different intermediate contexts that it induces would make it possible to distinguish $\underset{\rightarrow}{a} \setminus \uparrow\downarrow(\underset{\rightarrow}{c} / \underset{\leftarrow}{b})$from $\uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b}$.

But by using least symmetric relation $\mathcal{R}$ such that $\underset{\rightarrow}{a} \setminus \uparrow\downarrow(\underset{\rightarrow}{c} / \underset{\leftarrow}{b})\ \mathcal{R}\ \uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b}$ and $\underset{\rightarrow}{c} / \underset{\leftarrow}{b}\ \mathcal{R}\ \underset{\rightarrow}{a}\ (\uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) / \underset{\leftarrow}{b})$ and $\big(\underset{\rightarrow}{a} \setminus \uparrow\downarrow(\underset{\rightarrow}{c} / \underset{\leftarrow}{b})\big)\ \underset{\leftarrow}{b}\ \mathcal{R}\ \underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}$ and $\underset{\rightarrow}{c}\ \mathcal{R}\ \underset{\rightarrow}{c}$ and $(\cdot)\ \mathcal{R}\ (\cdot)$, we can prove that the two propositions are indistinguishable. The labeled bisimulation $\mathcal{R}$ shows how the inputs protected by the $\uparrow\downarrow$ double shifts are

treated lazily in establishing the equivalence: the proposition $\underset{\rightarrow}{c} / \underset{\leftarrow}{b}$ is $\mathcal{R}$-related to the context $\underset{\rightarrow}{a} \, (\uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) \, / \, \underset{\leftarrow}{b})$, for example.

7.1.2 *A simple up-to proof technique: Reflexivity*

As a slight enhancement of the above proof technique, we can consider a simple up-to technique: bisimilarity up to reflexivity. Let us call a relation $\mathcal{R}$ a labeled bisimulation *up to reflexivity* if $\mathcal{R}$ progresses to its reflexive closure, which we write as $\mathcal{R}^{=}$.

THEOREM 7.8. *If $\mathcal{R}$ is a labeled bisimulation up to reflexivity, then rewriting bisimilarity contains $\mathcal{R}$.*

Proof. Let $\mathcal{R}$ be a labeled bisimulation up to reflexivity. First, notice that the identity relation is a labeled bisimulation – each of the labeled bisimulation conditions is trivially true of the identity relation. Then, it follows from lemma 7.3 that $\mathcal{R}^{=}$, the reflexive closure of $\mathcal{R}$, is a labeled bisimulation. By theorem 7.6, we may conclude that rewriting bisimilarity contains $\mathcal{R}^{=}$ and hence $\mathcal{R}$. □

7.1.3 *Other properties of rewriting bisimilarity*

In addition to soundness and completeness of labeled bisimilarity with respect to rewriting bisimilarity, we also expect rewriting bisimilarity to be a (monoidal) congruence relation.

Rewriting bisimilarity is, indeed, an equivalence relation.

THEOREM 7.9. *Rewriting bisimilarity is reflexive, symmetric, and transitive.*

Proof. The identity relation on contexts can be shown to be a bisimulation, so rewriting bisimilarity is reflexive. Rewriting bisimilarity is symmetric by definition. The relation $\cong\cong$ can be shown to be a bisimulation, so rewriting bisimilarity is also transitive. □

At this point, we would like to prove, as a lemma, that $(\Delta_L \mathcal{R})$ is contained in some labeled bisimulation, for all contexts Δ_L and all labeled bisimulations $\mathcal{R}$. Ideally, the proof that proceeds by induction (or possibly coinduction), decomposing the context Δ_L and framing each antecedent of Δ_L onto the relation, one at a time. This would allow lemma 7.4 to be reused, and would also streamline other cases.

Unfortunately, such a proof has been elusive so far. So, instead, we will prove the following lemma by handling the context Δ_L all at once. The proof rehashes cases from lemma 7.4 and is not particularly enlightening beyond what was already presented there. For that reason, the proof is elided.

LEMMA 7.10. *If $\mathcal{R}$ is a labeled bisimulation, then, for each context Δ_L, there exists a labeled bisimulation that contains $(\Delta_L \mathcal{R})$.*

THEOREM 7.11. *If $\Omega_1 \cong \Delta_1$ and $\Omega_2 \cong \Delta_2$, then $\Omega_1 \, \Omega_2 \cong \Delta_1 \, \Delta_2$.*

Proof. Assume that $\Omega_1 \cong \Delta_1$ and $\Omega_2 \cong \Delta_2$. Notice that $\Omega_1\,\Omega_2\ (\Omega_1\cong)\ \Omega_1\,\Delta_2$ and that $\Omega_1\,\Delta_2\ (\cong\Delta_2)\ \Delta_1\,\Delta_2$.

Rewriting bisimilarity is a labeled bisimulation (theorem 7.2). According to lemma 7.10, there exists a labeled bisimulation that contains $(\Omega_1\cong)$. By theorem 7.6, rewriting bisimilarity therefore contains $(\Omega_1\cong)$. Using symmetric reasoning, rewriting bisimilarity must also contain $(\cong\Delta_2)$.

Applying these to the previous observation, $\Omega_1\,\Omega_2 \cong \Omega_1\,\Delta_2 \cong \Delta_1\,\Delta_2$. Because rewriting bisimilarity is transitive (theorem 7.9), we conclude that $\Omega_1\,\Omega_2 \cong \Delta_1\,\Delta_2$. □

Corollary 7.12. *Rewriting bisimilarity is a congruence.*

7.2 *Example: Rewriting bisimilarity and NFAs*

Recall from section 6.5.3 the conjecture that bisimilar NFA states have bisimilar encodings and vice versa. As outlined there, establishing this result will allow us to rephrase soundness and completeness of the NFA choreography in a properly stratified way: $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow_\Phi\cong \hat{q}'$ if, and only if, $q \xrightarrow{a}\sim q'$. More precisely, we will prove the following theorem.

Theorem 7.14. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ. Then $q \sim s$ if, and only if, $\hat{q} \cong \hat{s}$ for all states q and s.*

Before proving this statement, we need a few lemmas.

These results hold only because the formula-as-process ordered rewriting framework is focused; under an unfocused rewriting framework, $\hat{q}$ would admit rewritings, such as $\hat{q} \Longrightarrow \underset{\rightarrow}{\$} \setminus \underset{\rightarrow}{F}(q)$, and $\underset{\rightarrow}{a}\,\hat{q}$ would admit rewritings to contexts other than encodings of a-successors.

Lemma 7.13. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the alphabet Σ. Then:*

- *$\hat{q} \not\longrightarrow$ for all states q.*
- *If $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow\cong \hat{q}'$, then $\hat{q}'_a \cong \hat{q}'$ for some state q'_a that a-succeeds q.*
- *If $\underset{\rightarrow}{\$}\,\hat{q} \Longrightarrow\cong \underset{\rightarrow}{F}(s)$, then $q \in F$ if, and only if, $s \in F$.*

Proof. The first part can be proved by examining the encoding of an arbitrary state q.

To prove the second part, assume that $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow\cong \hat{q}'$. By inversion on the given trace, there are two cases: either *(i)* $\underset{\rightarrow}{a}\,\hat{q} \cong \hat{q}'$ or *(ii)* $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a \Longrightarrow\cong \hat{q}'$ for some state q'_a that a-succeeds q.

- Consider the case in which $\underset{\rightarrow}{a}\,\hat{q} \cong \hat{q}'$. Because the underlying NFA is well-formed (definition 2.1), q has at least one a-successor; let q'_a be one such successor. By definition of the encoding, $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a$. Because rewriting bisimilarity is reduction-closed (theorem 7.1), $\hat{q}'_a \cong\Longleftarrow \hat{q}'$. The first part of this lemma shows that states are encoded by propositions that do not reduce, and so we may conclude that, in fact, $\hat{q}'_a \cong \hat{q}'$.

- Consider the case in which $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a \Longrightarrow\cong \hat{q}'$ for some state q'_a that a-succeeds q. Because states are encoded by propositions that do not reduce, $\hat{q}'_a \cong \hat{q}'$ for some state q'_a that a-succeeds q, as required.

To prove the third part, we reason as in the preceding part and deduce that $\underset{\rightarrow}{F}(q) \cong \underset{\rightarrow}{F}(s)$; we conclude that $q \in F$ if, and only if, $s \in F$. □

THEOREM 7.14. *Let $\mathcal{A} = (Q, \Delta, F)$ be an NFA over the input alphabet Σ. Then $q \sim s$ if, and only if, $\hat{q} \cong \hat{s}$ for all states q and s.*

Proof. We shall show that NFA bisimilarity coincides with rewriting bisimilarity of encodings, proving each direction separately.

- To prove that bisimilar NFA states have bisimilar encodings – *i.e.*, that $q \sim s$ implies $\hat{q} \cong \hat{s}$ – we shall now show that the relation $\mathcal{R} = \{(\hat{q}, \hat{s}) \mid q \sim s\}$ is a labeled bisimulation up to reflexivity and, by theorem 7.8, is included in rewriting bisimilarity. Notice that $\mathcal{R}$ is, by definition, symmetric because NFA bisimilarity is symmetric (theorem 7.9).

  Immediate output bisimulation Assume that $\hat{q}\ \mathcal{R}\ \hat{s} = \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$; we must show that $\hat{q} \Longrightarrow(\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \hat{s}$. By definition of the encoding, $\hat{s}$ is a negative proposition and does not expose outputs. Therefore, $\underset{\leftarrow}{\Delta}'_L$ and $\underset{\rightarrow}{\Delta}'_R$ are empty and Δ' is $\hat{s}$. The required $\hat{q} \Longrightarrow(\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \hat{s}$ follows trivially.

  Immediate input bisimulation Assume that $\hat{q}\ \mathcal{R}\ \hat{s}$ and $\underset{\rightarrow}{\Delta}_L\ [\hat{s}]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$; we must show that $\underset{\rightarrow}{\Delta}_L\ \hat{q}\ \underset{\leftarrow}{\Delta}_R \Longrightarrow \mathcal{R}^{=}\ \Delta'$. Inversion of the input transition yields two cases.

  - Consider the case in which the input transition is $\underset{\rightarrow}{a}\,[\hat{s}] \longrightarrow \hat{s}'_a$, where state s is a-succeeded by s'_a. Because q and s are bisimilar, there must exist an a-successor of q, say q'_a, that is bisimilar to s'_a. By definition of the encoding, we thus have $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a$. So indeed, because q'_a and s'_a are bisimilar states, $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \mathcal{R}^{=}\ \hat{s}'_a$, as required.
  - Consider the case in which the input transition is $\underset{\rightarrow}{\$}\,[\hat{s}] \longrightarrow \underset{\rightarrow}{F}(s)$. Because q and s are bisimilar states, $\underset{\rightarrow}{F}(q) = \underset{\rightarrow}{F}(s)$. By definition of the encoding, $\underset{\rightarrow}{\$}\,\hat{q} \longrightarrow \underset{\rightarrow}{F}(q)$, and so, indeed, $\underset{\rightarrow}{\$}\,\hat{q} \Longrightarrow \mathcal{R}^{=}\ \underset{\rightarrow}{F}(s)$, as required.

  Reduction bisimulation Assume that $\hat{q}\ \mathcal{R}\ \hat{s} \longrightarrow \Delta'$. The reduction bisimulation property holds vacuously because states are encoded as propositions that do not reduce (lemma 7.13) – there is no Δ' such that $\hat{s} \longrightarrow \Delta'$.

  Emptiness bisimulation Assume that $\hat{q}\ \mathcal{R}\ \hat{s} = (\cdot)$. The emptiness bisimulation property also holds vacuously because states are encoded as propositions, not empty contexts.

- To prove the converse – that states with bisimilar encodings are themselves bisimilar – we shall now show that the relation $\mathcal{R} = \{(q, s) \mid \hat{q} \cong \hat{s}\}$, which relates states if they have rewriting-bisimilar encodings, is an NFA bisimulation and is therefore included in NFA bisimilarity.

Because rewriting bisimilarity is symmetric (theorem 7.9), so too is the relation $\mathcal{R}$. We must also prove that $\mathcal{R}$ satisfies the conditions of NFA bisimilarity.

Input bisimulation Let q and s be states with bisimilar encodings, and let q'_a be an a-successor of q; we must exhibit a state s'_a that a-succeeds s and has an encoding that is bisimilar to that of q'_a.

By definition of the encoding, $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a$. Because q and s have bisimilar encodings, the input bisimulation property allows us to deduce that $\underset{\rightarrow}{a}\,\hat{s} \Longrightarrow\!\cong \hat{q}'_a$. An appeal to lemma 7.13 provides exactly what is needed: a state s'_a that a-succeeds s and has an encoding bisimilar to that of q'_a.

Finality bisimulation Let q and s be states with bisimilar encodings, and assume that q is a final state; we must show that s is also a final state.

By definition of the encoding, $\underset{\rightarrow}{\$}\,\hat{q} \longrightarrow \underset{\rightarrow}{F}(q) = \underset{\rightarrow}{y}$. Because q and s have bisimilar encodings, it follows from the input bisimulation property that $\underset{\rightarrow}{\$}\,\hat{s} \Longrightarrow\!\cong \underset{\rightarrow}{F}(q)$. An appeal to lemma 7.13 allows us to conclude that s, like q, is a final state. □

7.3 *Example: Rewriting bisimilarity and binary counters*

For a further application of rewriting bisimilarity, we can revisit the binary counter. Recall from section 6.4.1 its object-oriented choreography:

$$\begin{aligned}
\Phi = \hat{e} &\triangleq (\hat{e} \bullet \hat{b}_1 \;/\; \underset{\leftarrow}{i}) \;\&\; (\underset{\leftarrow}{z} \;/\; \underset{\leftarrow}{d}) \\
\hat{b}_0 &\triangleq (\uparrow\!\downarrow \hat{b}_1 \;/\; \underset{\leftarrow}{i}) \;\&\; (\underset{\leftarrow}{d} \bullet \hat{b}'_0 \;/\; \underset{\leftarrow}{d}) \\
\hat{b}_1 &\triangleq (\underset{\leftarrow}{i} \bullet \hat{b}_0 \;/\; \underset{\leftarrow}{i}) \;\&\; (\hat{b}_0 \bullet \underset{\rightarrow}{s} \;/\; \underset{\leftarrow}{d}) \\
\hat{b}'_0 &\triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \;\&\; (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})
\end{aligned}$$

Also, recall that denotations were assigned directly to choreographed counters using the $\Omega \approx_{\mathrm{V}} n$, $\Omega \approx_{\mathrm{I}} n$, and $\Omega \approx_{\mathrm{D}} n$ judgments.

Intuitively, any two counters that have the same denotation ought to be indistinguishable. After all, the only two operations that we have on counters are increment and head-unary normalization (also known as decrement), and both of these are reflected in the denotation. For instance, both $\hat{e}\,\underset{\leftarrow}{i}\,\hat{b}_1$ and $\hat{e}\,\hat{b}_1\,\hat{b}_0\,\underset{\leftarrow}{i}$ denote the natural number 3, so any sequence of increments and decrements that we apply to these counters ought not to distinguish them.

We can state and prove that counters with equal denotations are bisimilar, and conversely, that bisimilar counters have equal denotations.

Theorem 7.16. *If either* (i) $\Omega \approx_{\mathrm{I}} n$ *and* $\Delta \approx_{\mathrm{I}} n'$ *or* (ii) $\Omega \approx_{\mathrm{D}} n$ *and* $\Delta \approx_{\mathrm{D}} n'$, *then* $\Omega \cong \Delta$ *if, and only if,* $n = n'$.

Before proving this theorem, recall the big-step adequacy of decrements for the choreography.

Corollary 6.14 (Adequacy of object-oriented choreography). *The following hold.*

Preservation *If* $\Omega \approx_{\mathrm{I}} n$ *and* $\Omega \longrightarrow_{\Phi} \Omega'$, *then* $\Omega' \approx_{\mathrm{I}} n$. *If* $\Omega \approx_{\mathrm{D}} n$ *and* $\Omega \longrightarrow_{\Phi} \Omega'$, *then* $\Omega' \approx_{\mathrm{D}} n$.

Big-step *If* $\Omega \approx_{\mathrm{D}} n$, *then:*

- $\Omega \Longrightarrow_{\Phi} \underset{\rightarrow}{z}$ *if, and only if,* $n = 0$;
- $\Omega \Longrightarrow_{\Phi} \Omega' \underset{\rightarrow}{s}$ *for some* Ω' *such that* $\Omega' \approx_{\mathrm{I}} n - 1$, *if* $n > 0$; *and*
- $\Omega \Longrightarrow_{\Phi} \Omega' \underset{\rightarrow}{s}$ *only if* $n > 0$ *and* $\Omega' \approx_{\mathrm{I}} n - 1$.

We also need to prove an easy lemma that characterizes the output and input transitions possible from binary counters.

Lemma 7.15.

- *If* $\Delta \approx_{\mathrm{I}} n$, *then:*
  - $\Delta = \underset{\leftarrow}{\Delta}'_L \, \Delta' \, \underset{\rightarrow}{\Delta}'_R$ *only if* $\underset{\rightarrow}{\Delta}'_L$ *and* $\underset{\rightarrow}{\Delta}'_R$ *are empty; and*
  - $\underset{\rightarrow}{\Delta}_L \, [\Delta] \, \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$ *only if* $\underset{\rightarrow}{\Delta}_L$ *is empty and either* $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{i}$ *or* $\underset{\leftarrow}{\Delta}_R = \underset{\leftarrow}{d}$.
- *If* $\Delta \approx_{\mathrm{D}} n$, *then:*
  - $\Delta = \underset{\leftarrow}{\Delta}'_L \, \Delta' \, \underset{\rightarrow}{\Delta}'_R$ *only if* $\underset{\rightarrow}{\Delta}'_L$ *is empty and either:*
    - $n = 0$ *and* $\underset{\rightarrow}{\Delta}'_R = \underset{\rightarrow}{z}$ *and* Δ' *is empty;*
    - $n > 0$ *and* $\underset{\rightarrow}{\Delta}'_R = \underset{\rightarrow}{s}$ *and* $\Delta' \approx_{\mathrm{I}} n - 1$; *or*
    - $\underset{\rightarrow}{\Delta}'_R$ *is empty; and*
  - $\underset{\rightarrow}{\Delta}_L \, [\Delta] \, \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$ *is impossible.*

Proof. By structural induction on the derivation of the given denotation. The second part relies on the first part in one case. □

Using this lemma, we may prove the correspondence between denotation and bisimilarity. It is especially interesting that the proof of this theorem is quite modular, relying heavily on the choreography's adequacy. We conjecture that this proof pattern will be useful in proving theorems about bisimilarities for other choreographies.

Theorem 7.16. *If either* (i) $\Omega \approx_{\mathrm{I}} n$ *and* $\Delta \approx_{\mathrm{I}} n'$ *or* (ii) $\Omega \approx_{\mathrm{D}} n$ *and* $\Delta \approx_{\mathrm{D}} n'$, *then* $\Omega \cong \Delta$ *if, and only if,* $n = n'$.

Proof. We prove each direction separately.

- To prove that states with equal denotations are bisimilar, consider the relation $\mathcal{R}$ given by

$$\begin{aligned}\mathcal{R} = \{(\Omega, \Delta) \mid \exists n \in \mathbb{N}.\, (\Omega \approx_{\mathrm{I}} n) \wedge (\Delta \approx_{\mathrm{I}} n)\} \\ \cup \{(\Omega, \Delta) \mid \exists n \in \mathbb{N}.\, (\Omega \approx_{\mathrm{D}} n) \wedge (\Delta \approx_{\mathrm{D}} n)\}\,.\end{aligned}$$

  We shall show that $\mathcal{R}$ progresses to its reflexive closure and then conclude, by theorem 7.8, that $\mathcal{R}$ is contained within rewriting bisimilarity.

  Immediate output bisimulation Assume that $\Omega \; \mathcal{R} \; \Delta = \underset{\leftarrow}{\Delta}'_L \, \Delta' \, \underset{\rightarrow}{\Delta}'_R$; we must show that $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R) \; \Delta$. Because Ω and Δ are $\mathcal{R}$-related, either: $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n$ for some natural number n; or $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n$ for some natural number n.

- Consider the case in which $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n$. Lemma 7.15 shows that $\underset{\leftarrow}{\Delta}'_L$ and $\underset{\rightarrow}{\Delta}'_R$ must both be empty. The required $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \Delta$ is then trivial.
- Consider the case in which $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n$. According to lemma 7.15, there are three cases. In all cases, $\underset{\leftarrow}{\Delta}'_L$ is empty.
  - Consider the case in which $\underset{\rightarrow}{\Delta}'_R$ is empty. Then $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \Delta$ is trivial.
  - Consider the case in which $n = 0$ and $\Delta = \underset{\rightarrow}{\Delta}'_R = \underset{\rightarrow}{z}$. According to the big-step adequacy of decrements for the object-oriented choreography (corollary 6.14), $\Omega \Longrightarrow \underset{\rightarrow}{z}$. It follows immediately that $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \underset{\rightarrow}{z} = \Delta$, as required.
  - Consider the case in which $n > 0$ and $\Delta = \Delta' \underset{\rightarrow}{s}$ with $\Delta' \approx_{\mathrm{I}} n-1$. According to the big-step adequacy of decrements for the choreography (corollary 6.14), $\Omega \Longrightarrow \Omega' \underset{\rightarrow}{s}$ for some Ω' such that $\Omega' \approx_{\mathrm{I}} n-1$. Therefore, $\Omega'\ \mathcal{R}\ \Delta'$. The required $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R}^{=} \underset{\rightarrow}{\Delta}'_R)\ \Delta' \underset{\rightarrow}{s} = \Delta$ follows.

Immediate input bisimulation Assume that $\Omega\ \mathcal{R}\ \Delta$ and $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$. There are several cases.

- Consider the case in which $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n$, for some natural number n. According to lemma 7.15, the input transition is either $[\Delta]\ \underset{\leftarrow}{i} \longrightarrow \Delta'$ or $[\Delta]\ \underset{\leftarrow}{d} \longrightarrow \Delta'$.
  - If the input transition $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$ is $[\Delta]\ \underset{\leftarrow}{i} \longrightarrow \Delta'$, then apply the $\underset{\leftarrow}{i}$-I rule to deduce that $\Omega\ \underset{\leftarrow}{i} \approx_{\mathrm{I}} n+1$ and $\Delta\ \underset{\leftarrow}{i} \approx_{\mathrm{I}} n+1$. Because $\Delta\ \underset{\leftarrow}{i} \longrightarrow \Delta'$, it follows from $\approx_{\mathrm{I}}$-preservation (corollary 6.14) that $\Delta' \approx_{\mathrm{I}} n+1$. We conclude that $\Omega\ \underset{\leftarrow}{i} \Longrightarrow \mathcal{R}\ \Delta'$, as required.
  - If the input transition $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$ is $[\Delta]\ \underset{\leftarrow}{d} \longrightarrow \Delta'$, then similar reasoning applies.
- Consider the case in which $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n$, for some n. By lemma 7.15, this input transition is impossible.

Reduction bisimulation Assume that $\Omega\ \mathcal{R}\ \Delta \longrightarrow \Delta'$. If $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n$ for some natural number n, then $\approx_{\mathrm{I}}$-preservation (corollary 6.14) yields $\Delta' \approx_{\mathrm{I}} n$; it immediately follows that $\Omega \Longrightarrow \mathcal{R}\ \Delta'$. Otherwise, if $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n$ for some natural number n, then $\approx_{\mathrm{D}}$-preservation similarly allows us to conclude that $\Omega \Longrightarrow \mathcal{R}\ \Delta'$.

Emptiness bisimulation This is vacuously true because $(\cdot)$ has no denotation under $\approx_{\mathrm{I}}$ and $\approx_{\mathrm{D}}$.

- To prove the converse, that bisimilar counters have equal denotations, we shall prove that

1. If $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n'$ and $\Omega \cong \Delta$, then $n = n'$.
2. If $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n'$ and $\Omega \cong \Delta$, then $n = n'$.

using a lexicographic induction, first on the denotation n and then on the inductive hypothesis used, with $1 < 2$.

1. Assume that $\Omega \approx_{\mathrm{D}} n$ and $\Delta \approx_{\mathrm{D}} n'$ and $\Omega \cong \Delta$.
   - Consider the case in which $n = 0$. By big-step adequacy of decrements (corollary 6.14), $\Omega \Longrightarrow \underset{\rightarrow}{z}$. Because Ω and Δ are bisimilar, $\Delta \Longrightarrow (\cong \underset{\rightarrow}{z})$ $\underset{\rightarrow}{z}$. According to big-step adequacy of decrements again (corollary 6.14), Δ eventually emits $\underset{\rightarrow}{z}$ only if its denotation is $n' = 0$, and so $n = 0 = n'$.
   - Consider the case in which $n > 0$. By big-step adequacy of decrements (corollary 6.14), $\Omega \Longrightarrow \Omega' \underset{\rightarrow}{s}$ for some Ω' such that $\Omega' \approx_{\mathrm{I}} n - 1$. Because Ω and Δ are bisimilar, $\Delta \Longrightarrow (\cong \underset{\rightarrow}{s})$ $\Omega' \underset{\rightarrow}{s}$; in other words, $\Delta \Longrightarrow \Delta' \underset{\rightarrow}{s}$ for some Δ' such that $\Omega' \cong \Delta'$. According to big-step adequacy of decrements again (corollary 6.14), Δ eventually emits $\underset{\rightarrow}{s}$ only if $n' > 0$ and $\Delta' \approx_{\mathrm{I}} n' - 1$. By the inductive hypothesis, it follows that $n - 1 = n' - 1$, and so $n = n'$ as required.
2. Assume that $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n'$ and $\Omega \cong \Delta$. By applying the $\underset{\leftarrow}{d}$-D rule, we may deduce that $\Omega \underset{\leftarrow}{d} \approx_{\mathrm{D}} n$ and $\Delta \underset{\leftarrow}{d} \approx_{\mathrm{D}} n'$. Moreover, because rewriting bisimilarity is a congruence (corollary 7.12), $\Omega \underset{\leftarrow}{d} \cong \Delta \underset{\leftarrow}{d}$. By part 1 of the inductive hypothesis, we conclude that $n = n'$, as required.

□

7.3.1 *A comment on atom directions and bisimilarity*

This theorem gives us the opportunity to remark on the interplay between atoms' directionality and rewriting bisimilarity.

Suppose that the formula-as-process ordered rewriting framework was designed without assigning direction to atoms. Instead of the directed atoms $\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{a}$, the framework would have only undirected atoms $\underline{a}$. The definition of rewriting bisimilarity would also be ever so slightly revised to use the undirected atoms. Rewriting bisimilarity would be the largest symmetric relation $\mathcal{R}$ to satisfy:

Output bisimulation If $\Omega \mathcal{R} \Longrightarrow \underline{\Delta}'_L \, \Delta' \, \underline{\Delta}'_R$, then $\Omega \Longrightarrow (\underline{\Delta}'_L \mathcal{R} \underline{\Delta}'_R)$ $\underline{\Delta}'_L \, \Delta' \, \underline{\Delta}'_R$.
Input bisimulation If $\underline{\Delta}_L \, \Omega \, \underline{\Delta}_R$ $(\underline{\Delta}_L \mathcal{R} \underline{\Delta}_R) \Longrightarrow \Delta'$, then $\underline{\Delta}_L \, \Omega \, \underline{\Delta}_R \Longrightarrow \mathcal{R} \, \Delta'$.

Unfortunately, this undirected notion of bisimilarity would be too fine. Without atom directions to distinguish messages intended as inputs from those intended as outputs, input messages could be incorrectly observed as outputs. In practice, this means that bisimilarity would rule out desirable equivalences.

Consider, for example, $\hat{e}$ and $\hat{e}\,\hat{b}_0$. These two counters have the same denotation – both represent 0. Nevertheless, in opposition to theorem 7.16 for directed atoms, $\hat{e}$ and $\hat{e}\,\hat{b}_0$ would *not* be bisimilar when using undirected atoms.

To see why, suppose, for the sake of deriving a contradiction, that $\hat{e} \cong \hat{e}\,\hat{b}_0$. Because $\hat{e}\,\hat{b}_0 \, \underline{d} \Longrightarrow \underline{z}\,\hat{b}'_0$, it follows from the input bisimulation property that

$\hat{e}\,\underline{d} \Longrightarrow\cong \underline{z}\,\hat{b}'_0$. There are only two contexts that can arise from $\hat{e}\,\underline{d}$, so either $\hat{e}\,\underline{d} \cong \underline{z}\,\hat{b}'_0$ or $\underline{z} \cong \underline{z}\,\hat{b}'_0$.

- The former is impossible because $\underline{z}\,\hat{b}'_0$ cannot produce $\underline{d}$ on the right (nor on the left) and so violates output bisimilarity.
- The latter is also impossible. It has an output of $\underline{z}$ on the left of $\underline{z}\,\hat{b}'_0$, from which the output bisimulation property yields $(\cdot) \cong \hat{b}'_0$. From the input bisimulation property, $\underline{a} \cong \hat{b}'_0\,\underline{a}$ follows, for any atom $\underline{a}$. And that violates output bisimulation because $\hat{b}'_0\,\underline{a}$, which does not reduce, cannot match the left output that $\underline{a}$ makes.

The key feature of this counterexample is that atoms' lack of direction means that the output bisimilarity condition also applies to atoms intended to act as inputs ($\underline{d}$ and $\underline{a}$, for instance). Just as atom directions were shown in section 6.1.2 to prevent a process from capturing a message it just sent, so do atom directions prevent *input* messages from being observed.

Part III

Concurrency as proof reduction

8
Singleton logic

Intuitionistic sequents are typically asymmetric: in an intuitionistic sequent $\Gamma \vdash A$, there are finitely many antecedents, all collected into the context Γ, yet there is only a single consequent, A.[1] We might naturally wonder if a greater degree of symmetry can be brought to sequents. Of course, classical sequents in calculi such as Gentzen's LK[2] are symmetric, but does there exist an *intuitionistic* logic whose sequent calculus presentation enjoys a similarly pleasant symmetry?

One approach might be to permit finitely many consequents, as in multiple-conclusion sequent calculi for intuitionistic logic,[3] but Steinberger[4] raises troubling concerns about the validity of meaning-theoretic explanations of such calculi, primarily that the meanings of the logical connectives are not properly independent but inextricably linked with the meaning of disjunction.

So, in this chapter, we will instead follow a dual path to symmetry and examine a restriction in which sequents have exactly one antecedent – no more and no less. We call this requirement the *single-antecedent restriction*; the sequent calculus to which it leads, the *singleton sequent calculus*; and the underlying logic, *singleton logic*. That such a severe restriction on the structure of sequents yields a well-defined, computationally useful logic is quite surprising.

Aside from motivations of symmetry, the single-antecedent restriction is sensible within each branch of the computational trinity[5] – proof theory, category theory, and type theory – as we sketch in section 8.1. This chapter will thereafter focus on the proof-theoretic consequences of the single-antecedent restriction.

Having fully motivated the single-antecedent restriction, we then proceed to section 8.2 where we derive the singleton sequent calculus by systematically applying the restriction to the intuitionistic ordered sequent calculus of chapter 3. (Not all of the ordered logical connectives will be able to survive the restriction, however. As we will explain, it is precisely the multiplicative connectives that are absent from singleton logic.)

[1] Or, at most one consequent if multiplicative falsehood is included (see, for example, section 3.4).

[2] Gentzen 1935.

[3] Maehara 1954; Kleene 1952.

[4] Steinberger 2011.

[5] Harper 2011.

To ensure that the resulting calculus properly defines the meaning of each connective by its inference rules, section 8.2.1 establishes the calculus's basic metatheory. Together, the cut elimination and identity elimination metatheorems identify the cut-free, identity-long proofs as verifications that form the foundation by exhibiting a subformula property.

Yet there are certainly other presentations of logics besides sequent calculi, so, in section 8.3, we develop a Hilbert-style axiomatization of singleton logic. This Hilbert system can also be viewed as a variant of the sequent calculus, so we dub it the *semi-axiomatic sequent calculus*.[6] An analysis of its basic metatheory (section 8.3.2) begins to suggest the basis of a Curry–Howard interpretation of semi-axiomatic sequent proofs as chains of well-typed, asynchronously communicating processes. Chapter 9 will be devoted to developing that observation more fully.

[6] DeYoung, Pfenning, and Pruiksma 2020.

Finally, section 8.4 briefly overviews several possible extensions to singleton logic, including a *sub*singleton extension that relaxes the single-antecedent restriction and permits an empty context.

8.1 *The single-antecedent restriction*

As sketched above, the *single-antecedent restriction* demands that each sequent contain exactly one antecedent, so that sequents are $A \vdash B$ instead of $\Gamma \vdash B$.

In addition to providing sequents with an elegant symmetry between antecedents and consequents, the single-antecedent restriction is a worthwhile object of investigation when viewed from the perspective of each branch of the computational trinity[7] – proof theory, category theory, and type theory:

[7] Harper 2011.

Proof theory In sequent calculi, antecedents are subject, either implicitly or explicitly, to structural properties, such as weakening, contraction, and exchange. For instance, antecedents in linear logic are subject to exchange, but neither weakening nor contraction; linear contexts thus form a commutative monoid over antecedents. Ordered logic goes further and rejects exchange; ordered contexts thus form a *non*commutative monoid.

Singleton logic is a natural object of investigation, precisely because it takes the idea of rejecting structural properties to its extreme. In adopting the single-antecedent restriction, singleton logic rejects the very idea that contexts have any structure whatsoever; there can be no binary operation to join contexts.

Category theory Each morphism in a category, $f : X \to Y$, has exactly one object – no more and no less – as its domain. Because sequents represent a kind of function, single antecedents are just as natural as single-object domains.

More specifically, in categorical semantics of sequent calculi, proofs are represented by the morphisms of a monoidal category, and so contexts of

several antecedents are packaged into a single domain object using the monoidal product:

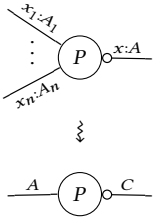
$$\ulcorner \mathcal{D} :: (A_1, A_2, \ldots, A_n \vdash B) \urcorner : \ulcorner A_1 \urcorner \otimes \ulcorner A_2 \urcorner \otimes \cdots \otimes \ulcorner A_n \urcorner \to \ulcorner B \urcorner$$

Because working in a monoidal category complicates matters, it is worthwhile to investigate whether there exists a sequent calculus whose categorical semantics uses no monoidal product. The single-antecedent restriction is exactly what results from these considerations, and the singleton sequent calculus will have a cleaner, more direct categorical semantics because of it.

Type theory In Caires, Pfenning, and Toninho's SILL type theory[8] based on intuitionistic linear logic, each well-typed process P acts as a client of multiple services $(A_i)_{i=1}^n$ along channels $(x_i)_{i=1}^n$, while simultaneously offering a service A of its own along a single channel x. Thus, networks of well-typed processes have a tree topology, as depicted in the neighboring display.

In data pipelines, the computational processes are arranged in a chain topology, with each process having exactly one upstream provider – no more and no less. To study pipelines, a "single-provider restriction" is needed – a type-theoretic analogue of the single-antecedent restriction.

[8] Caires and Pfenning 2010; Caires, Pfenning, and Toninho 2012; Toninho et al. 2013; Caires, Pfenning, and Toninho 2016.



8.2 *A sequent calculus for propositional singleton logic*

Having sketched proof-theoretic, category-theoretic, and type-theoretic reasons to investigate the single-antecedent restriction, we now turn to identifying a sequent calculus that satisfies that restriction.

ONE APPROACH to constructing a singleton sequent calculus is to take the intuitionistic ordered sequent calculus of chapter 3, apply the single-antecedent restriction to each rule's sequents, and solve the constraints that that restriction imposes.

For instance, consider the ordered cut rule (see neighboring display). For the first premise to satisfy the single-antecedent restriction, the finitary context Ω must be exactly a single antecedent, A. Because the second premise already contains the antecedent B, the contexts Ω'_L and Ω'_R must also be empty. After these revisions, the rule contains only well-formed singleton sequents and is a candidate for inclusion in the singleton sequent calculus.

We could equally well justify this new cut rule by first principles, as it expresses the composition of two well-formed singleton proofs. But the above method of considering the constraints imposed by the single-antecedent restriction is a straightforward, mechanical way ahead for the other inference rules. For example, singleton sequent calculus rules for additive disjunction may also be constructed in this way (see fig. 8.2). Rules for the other additive connectives ($\&$, $\top$, and $\mathbf{0}$) can be constructed, too, but we will momentarily postpone displaying them.

$$\frac{\Omega \vdash B \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^B$$

$$\vdots$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B$$

Figure 8.1: Deriving the singleton sequent calculus's cut rule from the corresponding ordered sequent calculus rule

Ordered sequent calculus *Singleton sequent calculus*

$$\frac{\Omega \vdash B_1}{\Omega \vdash B_1 \oplus B_2}\ \oplus\text{R}_1 \quad \frac{\Omega \vdash B_2}{\Omega \vdash B_1 \oplus B_2}\ \oplus\text{R}_2 \qquad \frac{A \vdash B_1}{A \vdash B_1 \oplus B_2}\ \oplus\text{R}_1 \quad \frac{A \vdash B_2}{A \vdash B_1 \oplus B_2}\ \oplus\text{R}_2$$

$$\frac{\Omega'_L\ B_1\ \Omega'_R \vdash C \quad \Omega'_L\ B_2\ \Omega'_R \vdash C}{\Omega'_L\ (B_1 \oplus B_2)\ \Omega'_R \vdash C}\ \oplus\text{L} \rightsquigarrow \frac{B_1 \vdash C \quad B_2 \vdash C}{B_1 \oplus B_2 \vdash C}\ \oplus\text{L}$$

Figure 8.2: Deriving the singleton sequent calculus rules for additive disjunction from the corresponding ordered sequent calculus rules

However, not all ordered logical connectives fare as well under the single-antecedent restriction as the additive connectives do. In particular, the multiplicative connectives do not have analogues in singleton logic. Consider, for example, left-handed implication and its right rule (see neighboring figure). The finitary context Ω must be replaced with a single antecedent, A, if the rule's conclusion is to be a well-formed singleton sequent. Now the revised rule's conclusion is well-formed, but its premise is not.

From a category-theoretic perspective, it would be quite natural to rewrite the premise using ordered conjunction so that the two antecedents are packaged together as one. However, from a proof-theoretic perspective, this rule is not suitable – with this rule, the meaning of left-handed implication depends on the meaning of another connective, namely multiplicative conjunction. As a practical consequence, the subformula property and related cut elimination theorem would fail to hold if the singleton sequent calculus adopted this rule.

$$\frac{B_1\ \Omega \vdash B_2}{\Omega \vdash B_1 \setminus B_2}\ \setminus\text{R}$$

$$\vdots$$

$$\frac{B_1\ A \vdash B_2}{A \vdash B_1 \setminus B_2}\ \setminus\text{R}?$$

$$\vdots$$

$$\frac{B_1 \bullet A \vdash B_2}{A \vdash B_1 \setminus B_2}\ \setminus\text{R}?$$

Figure 8.3: A failed attempt at constructing a right rule for left-handed implication

In attempting to construct singleton sequent calculus rules for left-handed implication, the fundamental problem is that the $\setminus$R rule introduces an additional antecedent to a context that is, and must remain, a singleton. Changing the size of the context by introducing, or sometimes removing, antecedents is an essential characteristic of multiplicative connectives, and so the multiplicative connectives, by their very nature, cannot appear in singleton logic.

Figure 8.4 presents the complete set of rules for propositional singleton logic's sequent calculus.

Although the propositions of singleton logic are exactly the additive propositions of ordered logic, singleton logic is *not* the additive fragment of ordered logic. For instance, the sequent $A\ B \vdash \top$ is provable in the additive fragment of ordered logic, but it is not even a well-formed sequent in the singleton sequent calculus, for the simple reason that it violates the single-antecedent restriction.

That said, singleton logic only differs from the additive fragment of ordered logic in its treatment of $\mathbf{0}$ and $\top$ – the $\mathbf{0}$,$\top$-free fragment of singleton logic coincides exactly with the $\mathbf{0}$,$\top$-free, additive fragment (that is, the $\oplus$,&-fragment) of ordered logic. A simple structural induction proves this:

Theorem 8.1. *If $\Omega \vdash B$ in the $\oplus$,&-fragment of the ordered sequent calculus, then there exists a proposition A such that $\Omega = A$ and $A \vdash B$ in the $\oplus$,&-fragment of the singleton sequent calculus.*

In substructural logics and systems built upon them, the logical constants $\mathbf{0}$ and $\top$ are often problematic because they indiscriminately consume any

$$\textsc{propositions} \quad A, B, C ::= a \mid A \oplus B \mid \mathbf{0} \mid A \,\&\, B \mid \top$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{A \vdash B_1}{A \vdash B_1 \oplus B_2}\ \oplus\textsc{r}_1 \qquad \frac{A \vdash B_2}{A \vdash B_1 \oplus B_2}\ \oplus\textsc{r}_2 \qquad \frac{B_1 \vdash C \quad B_2 \vdash C}{B_1 \oplus B_2 \vdash C}\ \oplus\textsc{l}$$

$$\text{(no } \mathbf{0}\textsc{r} \text{ rule)} \qquad \frac{}{\mathbf{0} \vdash C}\ \mathbf{0}\textsc{l}$$

$$\frac{A \vdash B_1 \quad A \vdash B_2}{A \vdash B_1 \,\&\, B_2}\ \&\textsc{r} \qquad \frac{B_1 \vdash C}{B_1 \,\&\, B_2 \vdash C}\ \&\textsc{l}_1 \qquad \frac{B_2 \vdash C}{B_1 \,\&\, B_2 \vdash C}\ \&\textsc{l}_2$$

$$\frac{}{A \vdash \top}\ \top\textsc{r} \qquad \text{(no } \top\textsc{l} \text{ rule)}$$

Figure 8.4: A sequent calculus for propositional singleton logic

and all resources placed in front of them.[9] Interestingly, because the single-antecedent restriction exactly constrains the resources that those logical constants may consume, working in singleton logic is one possible way to sanitize $\mathbf{0}$ and $\top$.

[9] Cervesato, Hodas, et al. 2000; Schack-Nielsen and Schürmann 2008.

Symmetry was one of the motivations behind examining the single-antecedent restriction and the singleton logic that results from it. With a sequent calculus for singleton logic, we can now make that symmetry precise. For instance, the $\&\textsc{r}$ rule can be exactly obtained from the $\oplus\textsc{l}$ rule – and *vice versa* – by reversing the turnstiles and replacing the $\oplus$ connective with $\&$:

$$\frac{A_1 \vdash B \quad A_2 \vdash B}{A_1 \oplus A_2 \vdash B}\ \oplus\textsc{l} \quad \leftrightsquigarrow \quad \frac{A_1 \dashv B \quad A_2 \dashv B}{A_1 \,\&\, A_2 \dashv B}\ \&\textsc{r}$$

Applying this transformation to the other rules results in similar symmetric pairs. The cut even maps to itself.

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B \quad \leftrightsquigarrow \quad \frac{A \dashv B \quad B \dashv C}{A \dashv C}\ \textsc{cut}^B$$

To make this automorphism formal, we define an involution, $(-)^\perp$, on propositions (see the adjacent figure), presuming that the involution of an atom is again an atom. With this involution, it is relatively straightforward to state and prove symmetry:

$$(a)^\perp = a^\perp \qquad (a^\perp)^\perp = a$$
$$(A \oplus B)^\perp = A^\perp \,\&\, B^\perp$$
$$(\mathbf{0})^\perp = \top$$
$$(A \,\&\, B)^\perp = A^\perp \oplus B^\perp$$
$$(\top)^\perp = \mathbf{0}$$

Figure 8.5: An involution on propositions

Theorem 8.2. *$A \vdash B$ if and only if $B^\perp \vdash A^\perp$.*

Proof. The left-to-right direction can be proved by structural induction on the derivation of $A \vdash B$. The converse follows immediately, because $(-)^\perp$ is an involution. □

Notice that the $(-)^\perp$ involution is the additive fragment of the involution commonly used in one-sided sequent calculi for classical linear logic. In this

sense, singleton logic exhibits the same symmetries as classical logic, but in an intuitionistic setting.

8.2.1 *Metatheory: Cut elimination and identity expansion*

The rules shown in fig. 8.4 certainly have the appearance of sequent calculus rules, but do they truly constitute a well-defined sequent calculus? Most peculiarly, the singleton sequent calculus has no implication connective that internalizes the underlying hypothetical judgment. Can such a calculus possibly be well-defined?

Because it coincides exactly with a fragment of the ordered sequent calculus (theorem 8.1), the singleton sequent calculus is indeed well-defined. However, for our subsequent development, it will prove useful to examine the singleton sequent calculus's metatheory, especially cut elimination, natively.

In the tradition of Gentzen, Dummett, and Martin-Löf,[10] a sequent calculus is well-defined if it rests on the solid foundation of a verificationist meaning-explanation. That is, the meaning of each logical connective must be given entirely by its right (and left) inference rules, and those rules must exist in harmony with the left rules.

[10] Gentzen 1935; Dummett 1976; Martin-Löf 1983.

A *verification*, then, is a proof that relies only on the right and left inference rules and the $\textsc{id}^p$ rule for propositional variables p – stated differently, verifications may not contain instances of the cut or general $\textsc{id}^A$ rules.

If every proof has a corresponding verification, then we can be sure that neither the cut nor id rules play any role in defining the logical connectives.

For this program to succeed, we need to be sure that for every proof there is a corresponding verification – in this sense, the usual cut elimination metatheorem states a weak normalization result.

Theorem 8.4 (Cut elimination). *If a proof of $A \vdash C$ exists, then there exists a* cut-free *proof of $A \vdash C$.*

As usual, the cut elimination theorem may be proved by a straightforward induction on the structure of the given proof, provided that a cut principle for cut-free proofs is admissible:

Lemma 8.3 (Admissibility of cut). *If cut-free proofs of $A \vdash B$ and $B \vdash C$ exist, then there exists a* cut-free *proof of $A \vdash C$.*

Before proceeding to this lemma's proof, it is worth emphasizing a subtle distinction between the singleton sequent calculus's primitive cut rule and the admissible cut principle that this lemma establishes.

To be completely formal, we could treat cut-freeness as an extrinsic, Curry-style property of proofs[11] and indicate cut-freeness by decorating the turnstile, so that $A \vdash^{\mathrm{cf}} C$ denotes a cut-free proof of $A \vdash C$. The admissible cut

[11] Contrast this with a separate, intrinsically cut-free sequent calculus in the style of Church (Pfenning 2008).

principle stated in lemma 8.3 could then be expressed as the rule

$$\dfrac{A \vdash^{\mathrm{cf}} B \quad B \vdash^{\mathrm{cf}} C}{A \vdash^{\mathrm{cf}} C}\ \textsc{a-cut}^{B},$$

with the dotted line indicating that it is an admissible, not primitive, rule. Writing it in this way emphasizes that proving lemma 8.3 amounts to defining a meta-level function that takes cut-free proofs of $A \vdash B$ and $B \vdash C$ and produces a *cut-free* proof of $A \vdash C$. Contrast this with the primitive CUT rule of the singleton sequent calculus, which forms a (cut-full) proof of $A \vdash C$ from (potentially cut-full) proofs of $A \vdash B$ and $B \vdash C$.

$$\dfrac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^{B}$$

From here on, however, we won't bother to be quite so pedantic, instead often omitting the turnstile decoration on cut-free proofs with the understanding that the admissible A-CUT rule may only be applied to cut-free proofs.

WITH THAT CLARIFICATION out of the way, we are finally ready to prove the admissibility of cut lemma.

LEMMA 8.3 (Admissibility of cut). *If cut-free proofs of $A \vdash B$ and $B \vdash C$ exist, then there exists a* cut-free *proof of $A \vdash C$.*

Proof. Just as in the proof of admissibility of cut for the ordered sequent calculus (lemma 3.1), we use a standard lexicographic structural induction, first on the structure of the cut formula, and then on the structures of the given proofs.

As usual, the cases can be classified into three categories: principal cases, identity cases, and commutative cases.

Principal cases As usual, the principal cases pair a proof ending in a right rule together with a proof ending in a corresponding left rule. One such principal case is:

$$\dfrac{\dfrac{\dfrac{\mathcal{D}_1}{A \vdash B_1}}{A \vdash B_1 \oplus B_2}\ \oplus\mathrm{R}_1 \qquad \dfrac{\dfrac{\mathcal{E}_1}{B_1 \vdash C} \quad \dfrac{\mathcal{E}_2}{B_2 \vdash C}}{B_1 \oplus B_2 \vdash C}\ \oplus\mathrm{L}}{A \vdash C}\ \textsc{a-cut}^{B_1 \oplus B_2}$$

$$=$$

$$\dfrac{\dfrac{\mathcal{D}_1}{A \vdash B_1} \quad \dfrac{\mathcal{E}_1}{B_1 \vdash C}}{A \vdash C}\ \textsc{a-cut}^{B_1}$$

Notice that the interaction between proofs here is synchronous – the case is resolved by appealing to the inductive hypothesis at a smaller cut formula but also smaller proofs.

Identity cases In the identity cases, one of the proofs is the ID rule alone. For example:

$$\dfrac{\dfrac{}{A \vdash A}\ \textsc{id}^{A} \quad \dfrac{\mathcal{E}}{A \vdash C}}{A \vdash C}\ \textsc{a-cut}^{A} \quad = \quad \dfrac{\mathcal{E}}{A \vdash C}.$$

Commutative cases As in the proof of ordered logic's admissible cut principle (lemma 3.1), the commutative cases are those in which one of the proofs ends by introducing a side formula.

As an example, one right-commutative case pairs a proof of $A \vdash B$ with a proof of $B \vdash C_1 \oplus C_2$ ending in the $\oplus\text{R}_1$ rule:

$$\dfrac{\mathcal{D}\quad \dfrac{\mathcal{E}_1}{B \vdash C_1}}{\dfrac{A \vdash B \qquad B \vdash C_1 \oplus C_2}{A \vdash C_1 \oplus C_2}\ \oplus\text{R}_1}\ \text{A-CUT}^B \quad = \quad \dfrac{\dfrac{\mathcal{D}}{A \vdash B}\quad \dfrac{\mathcal{E}_1}{B \vdash C_1}}{\dfrac{A \vdash C_1}{A \vdash C_1 \oplus C_2}\ \oplus\text{R}_1}\ \text{A-CUT}^B$$

Unlike in ordered logic, there can be no right-commutative cases involving left rules because the cut formula is the only antecedent in the sequent $B \vdash C$. In this way, the symmetry of singleton sequents is manifest even in proving the admissibility of cut. □

With the admissibility of cut established, we can finally prove cut elimination for the singleton sequent calculus.

Theorem 8.4 (Cut elimination). *If a proof of $A \vdash C$ exists, then there exists a* cut-free *proof of $A \vdash C$.*

Proof. By structural induction on the proof of $A \vdash C$, appealing to the admissibility of cut (lemma 8.3) when encountering a CUT rule.

If we display the inductive hypothesis as an admissible rule, then the crucial case in the proof of cut elimination is resolved as follows.

$$\dfrac{\dfrac{\mathcal{D}_1}{A \vdash B}\quad \dfrac{\mathcal{D}_2}{B \vdash C}\ \text{CUT}^B}{\dfrac{A \vdash C}{A \vdash^{\text{cf}} C}\ \text{CE}} \quad = \quad \dfrac{\dfrac{\dfrac{\mathcal{D}_1}{A \vdash B}\ \text{CE}}{A \vdash^{\text{cf}} B}\quad \dfrac{\dfrac{\mathcal{D}_2}{B \vdash C}\ \text{CE}}{B \vdash^{\text{cf}} C}\ \text{A-CUT}^B}{A \vdash^{\text{cf}} C}$$

All other cases are handled compositionally.

This cut elimination proof amounts to defining a meta-level function for normalizing proofs to cut-free form. □

In addition to cut elimination, we can also prove identity elimination. An identity-long proof is one in which all applications of the ID occur at propositional variables. Identity elimination – a slight misnomer – transforms a proof into an identity-long proof of the same sequent by replacing instances of the ID^A rule with an identity-long proof of $A \vdash A$.[12]

[12] Identity elimination is a slight misnomer because instances of the ID rule at propositional variables will remain.

Lemma 8.5 (Admissibility of identity). *For all propositions A, an identity-long proof of $A \vdash A$ exists. Moreover, this proof is cut-free.*

Proof. As usual, by induction on the structure of the proposition A. □

Theorem 8.6 (Identity elimination). *If a proof of $A \vdash C$ exists, then there exists an* identity-long *proof of $A \vdash C$. Moreover, if the given proof is cut-free, so is the identity-long proof.*

Proof. As usual, by structural induction on the proof of $A \vdash C$. □

8.3 *A semi-axiomatic sequent calculus for singleton logic*

Sequent calculi are not the only way to present logics, so in this section we also consider a Hilbert-style axiomatization of singleton logic, which can be viewed as a sequent calculus variant that we dub the *semi-axiomatic sequent calculus.*[13] Our interest in a semi-axiomatic sequent calculus for singleton logic is not taxonomic, however. Rather, over the course of the next chapter and a half, we shall see that normalization of semi-axiomatic sequent proofs serves as the basis of a Curry–Howard isomorphism with chains of asynchronously communicating processes.

[13] DeYoung, Pfenning, and Pruiksma 2020.

In a sequent calculus, the meaning of a connective is given by its right and left inference rules. Hilbert-style axiomatizations, on the other hand, strive to use as few rules of inference as possible, with the meaning of a connective instead given by a small collection of axiom schemas.

The term 'axiom schema' is often interpreted narrowly to mean only categorical judgments like $\vdash A \supset B \supset A \wedge B$, not hypothetical judgments like $\Gamma, A, B \vdash A \wedge B$ adopted as zero-premise rules of inference. Consequently, Hilbert-style axiomatizations usually rely heavily on implication and a *modus ponens* rule to effect the meanings of the logical connectives.

$$\frac{\vdash A \supset B \quad \vdash A}{\vdash B}\ \textsc{mp}$$

Figure 8.6: *Modus ponens* for a Hilbert-style axiomatization of intuitionistic logic

However, as explained in section 8.2, singleton logic does not enjoy the luxury of an implication connective. So a true Hilbert-style axiomatization that relies on a *modus ponens* rule as its only inference rule is not possible. Instead, for a semi-axiomatic sequent calculus of singleton logic, we must content ourselves with a broad interpretation of the term 'axiom schema' that encompasses zero-premise rules that derive hypothetical judgments.

To construct a semi-axiomatic sequent calculus for singleton logic, we will ask, in turn, whether each sequent calculus rule can be reduced to a schema.

First, consider the judgmental rules, id and cut, for the identity and cut principles (see neighboring display). With zero premises, the id rule itself is already an axiom schema and can be adopted directly in singleton logic's semi-axiomatic sequent calculus.

$$\frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B$$

The cut rule is not so accommodating. As a rule for composing proofs, the cut rule serves a similar purpose to the traditional *modus ponens* rule. Just as *modus ponens* cannot be reduced to an axiom schema, so must cut remain a rule of inference. Moreover, because singleton logic has no implication connective, the rule's hypothetical judgments cannot even be simplified to categorical judgments. Therefore, the cut rule is adopted wholesale in the semi-axiomatic sequent calculus.

Next, consider the sequent calculus's $\oplus\text{R}_1$ inference rule. Using the id axiom schema, we can obtain a zero-premise derived rule from $\oplus\text{R}_1$:

$$\frac{\dfrac{}{A_1 \vdash A_1}\ \textsc{id}}{A_1 \vdash A_1 \oplus A_2}\ \oplus\text{R}_1 \quad \leftrightsquigarrow \quad \frac{}{A_1 \vdash A_1 \oplus A_2}\ \oplus\text{R}'_1$$

$$\textsc{propositions} \quad A, B, C ::= a \mid A \oplus B \mid \mathbf{0} \mid A \,\&\, B \mid \top$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{}{A_1 \vdash A_1 \oplus A_2}\ \oplus\textsc{r}'_1 \qquad \frac{}{A_2 \vdash A_1 \oplus A_2}\ \oplus\textsc{r}'_2 \qquad \frac{A_1 \vdash C \quad A_2 \vdash C}{A_1 \oplus A_2 \vdash C}\ \oplus\textsc{l}$$

$$\text{(no } \mathbf{0}\textsc{r} \text{ rule)} \qquad \frac{}{\mathbf{0} \vdash C}\ \mathbf{0}\textsc{l}$$

$$\frac{A \vdash C_1 \quad A \vdash C_2}{A \vdash C_1 \,\&\, C_2}\ \&\textsc{r} \qquad \frac{}{C_1 \,\&\, C_2 \vdash C_1}\ \&\textsc{l}'_1 \qquad \frac{}{C_1 \,\&\, C_2 \vdash C_2}\ \&\textsc{l}'_2$$

$$\frac{}{A \vdash \top}\ \top\textsc{r} \qquad \text{(no } \top\textsc{l} \text{ rule)}$$

Figure 8.8: A semi-axiomatic for singleton logic

Moreover, by combining this new $\oplus\textsc{r}'_1$ axiom schema with CUT, we can recover the original $\oplus\textsc{r}_1$ rule as a derived rule:

$$\frac{A \vdash B_1 \quad \dfrac{}{B_1 \vdash B_1 \oplus B_2}\ \oplus\textsc{r}'_1}{A \vdash B_1 \oplus B_2}\ \textsc{cut} \qquad \leftrightsquigarrow \qquad \frac{A \vdash B_1}{A \vdash B_1 \oplus B_2}\ \oplus\textsc{r}_1$$

Together, these two observations suggest that $\oplus\textsc{r}'_1$ be adopted as an axiom schema in the semi-axiomatic sequent calculus for singleton logic. A symmetric $\oplus\textsc{r}'_2$ axiom schema should be adopted, too.

What about the sequent calculus's $\oplus\textsc{l}$ rule (see neighboring display)? Can it also be reduced to an axiom schema? Once again, singleton logic's lack of an implication connective prevents us from even simplifying the $\oplus\textsc{l}$ rule's hypothetical judgments to categorical judgments. Like CUT, the sequent calculus's $\oplus\textsc{l}$ rule is thus adopted wholesale in singleton logic's semi-axiomatic sequent calculus. Including the additive $\oplus\textsc{l}$ rule as a primitive rule of inference is perhaps not unexpected. It is consistent with Hilbert-style axiomatizations of linear logic,[14] which include an adjunction rule – essentially the linear sequent calculus's $\&\textsc{r}$ rule – to effect the additive behavior that linear implication and its multiplicative *modus ponens* rule cannot.

$$\frac{B_1 \vdash C \quad B_2 \vdash C}{B_1 \oplus B_2 \vdash C}\ \oplus\textsc{l}$$

[14] Avron 1988.

The treatment of additive conjunction is dual to that of $A_1 \oplus A_2$: The sequent calculus's $\&\textsc{r}$ rule will be adopted wholesale, and $\&\textsc{l}'_1$ and $\&\textsc{l}'_2$ axiom schemas will be derived from the sequent calculus's $\&\textsc{l}_1$, $\&\textsc{l}_2$, and ID rules. And finally, $\mathbf{0}$ and $\top$ are treated as the nullary analogues of those of the binary $\oplus$ and $\&$ connectives, respectively.

Figure 8.8 summarizes this semi-axiomatic sequent calculus for singleton logic.

$$\frac{A \vdash C_1 \quad A \vdash C_2}{A \vdash C_1 \,\&\, C_2}\ \&\textsc{r}$$

$$\frac{}{C_1 \,\&\, C_2 \vdash C_1}\ \&\textsc{l}'_1 \qquad \frac{}{C_1 \,\&\, C_2 \vdash C_2}\ \&\textsc{l}'_2$$

Figure 8.7: Semi-axiomatic sequent calculus rules for additive conjunction from singleton logic

THE SEMI-AXIOMATIC sequent calculus of fig. 8.8 shares many rules with the singleton sequent calculus (fig. 8.4). In fact, it is a variant in which each connective's non-invertible rules have been replaced with zero-premise rules.

$$\textsc{propositions} \quad A ::= a \mid \oplus_{\ell \in L}\{\ell : A_\ell\} \mid \&_{\ell \in L}\{\ell : A_\ell\}$$

$$\textsc{proof terms} \quad P ::= P_1 \diamond P_2 \mid \leftrightarrow \mid \underset{\rightarrow}{k} \mid \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \underset{\leftarrow}{k}$$

Figure 8.9: Proof terms for a labeled, n-ary variant of the semi-axiomatic sequent calculus of fig. 8.8

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \leftrightarrow : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon\ A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon\ A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underset{\leftarrow}{k} : C_k}\ \&\textsc{l}'$$

From this observation, we conjecture that every intuitionistic sequent calculus has a corresponding semi-axiomatic sequent calculus. But, in this document, we are only interested in the semi-axiomatic sequent calculus for singleton logic, so we do not pursue this conjecture further.

This semi-axiomatic variant being so closely related to the sequent calculus, we should seek to prove that it enjoys the usual sequent calculus metatheorems – cut elimination and identity expansion. Strictly speaking, however, cut elimination does not hold for the semi-axiomatic sequent calculus. As a concrete counterexample, there is no cut-free semi-axiomatic proof of the sequent $a_2 \vdash a_1 \oplus (a_2 \oplus a_3)$, even though the same sequent is provable using cut:

$$\frac{\dfrac{}{a_2 \vdash a_2 \oplus a_3}\ \oplus\textsc{r}_1' \qquad \dfrac{}{a_2 \oplus a_3 \vdash a_1 \oplus (a_2 \oplus a_3)}\ \oplus\textsc{r}_2'}{a_2 \vdash a_1 \oplus (a_2 \oplus a_3)}\ \textsc{cut}$$

Although cut elimination does not hold, normal forms nevertheless exist. Normal semi-axiomatic proofs will contain cuts, but those cuts will have a particular, analytic form. In other words, although full cut elimination does not hold, elimination of *non-analytic* cuts does.

8.3.1 *A proof term assignment for the semi-axiomatic sequent calculus*

Before presenting a proof of non-analytic cut elimination, we will take a moment to introduce a proof term assignment for the semi-axiomatic sequent calculus. These proof terms will be a convenient, succinct notation with which to describe the elimination procedure. To keep the proof terms compact, we will also take this opportunity to introduce labeled, n-ary forms of additive disjunction and conjunction.

Figure 8.9 presents a labeled, n-ary generalization of singleton logic's semi-axiomatic sequent calculus, equipped with proof terms. Individual labels ℓ and k are drawn from an unspecified universe of labels, and the metavariable L is used for index sets of labels. The labeled, n-ary proposition $\oplus_{\ell \in L}\{\ell : A_\ell\}$

generalizes binary additive disjunction, $A \oplus B$, and, because the label set L may even be empty, it also generalizes additive falsehood, $\mathbf{0}$. Likewise, $\&_{\ell \in L}\{\ell : A_\ell\}$ generalizes both $A \,\&\, B$ and $\top$.

Because the CUT rule serves to compose two proofs of compatible sequents, the proof term $P_1 \diamond P_2$ was chosen for its suggestion of function composition, $f_2 \circ f_1$.[15] The proof term $\leftrightarrow$ is used for the ID rule. Because of their similar structure, the $\oplus\text{R}'$ and $\&\text{L}'$ rules are assigned the similar proof terms $\underset{\rightarrow}{k}$ and $\underset{\leftarrow}{k}$; the direction of the underlying arrow distinguishes them. Similarly, the $\oplus$L and $\&$R rules are assigned the proof terms $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ and $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$. All of these terms serve as variable-free combinators.

[15] Notice that the order of composition in the $P_1 \diamond P_2$ term matches the order of premises in the CUT rule, but is opposite the order traditionally used for function composition.

8.3.2 *Non-analytic cut elimination for the semi-axiomatic sequent calculus*

With proof terms in hand, we can now return to our goal of establishing a *non-analytic* cut elimination theorem for singleton logic's semi-axiomatic sequent calculus.

The cut elimination procedure will normalize a semi-axiomatic proof so that any remaining cuts are analytic, specifically of the forms $\underset{\leftarrow}{k} \diamond P$ or $P \diamond \underset{\rightarrow}{k}$. As shown in the neighboring display, cuts of these forms are analytic because the cut formula is a subformula of the conclusion sequent.

$$\dfrac{\dfrac{(k \in L)}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash \underset{\leftarrow}{k} : A_k}\,\&\text{L}' \qquad A_k \vdash P : C}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash \underset{\leftarrow}{k} \diamond P : C}\,\text{CUT}^{A_k}$$

and

$$\dfrac{A \vdash P : C_k \qquad \dfrac{(k \in L)}{C_k \vdash \underset{\rightarrow}{k} : \bigoplus_{\ell \in L}\{\ell : C_\ell\}}\,\oplus\text{R}'}{A \vdash P \diamond \underset{\rightarrow}{k} : \bigoplus_{\ell \in L}\{\ell : C_\ell\}}\,\text{CUT}^{C_k}$$

We say that a term is *normal* if it contains only cuts of these analytic forms; the normal terms are generated by the following grammar.

$$\begin{aligned} N, M ::= {} & \leftrightarrow \mid N \diamond \underset{\rightarrow}{k} \mid \underset{\leftarrow}{k} \diamond N \\ & \mid \underset{\rightarrow}{k} \mid \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \\ & \mid \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \mid \underset{\leftarrow}{k} \end{aligned}$$

In other words, normality is an extrinsic property of terms that is judged by membership in the above grammar.

Non-analytic cut elimination then amounts to proof term normalization:

THEOREM 8.8 (Non-analytic cut elimination). *If $A \vdash P : C$, then $A \vdash N : C$ for some normal term N.*

Just like the sequent calculus's cut elimination result (theorem 8.4), this theorem can be proved by a straightforward structural induction, this time on the given term, P. First, however, we need the admissibility of non-analytic cut as a lemma:

LEMMA 8.7 (Admissibility of non-analytic cut). *If $A \vdash N : B$ and $B \vdash M : C$, then $A \vdash N' : C$ for some normal term N'.*

Proof. As with lemma 8.3 for the sequent calculus, this lemma states the admissibility of a cut principle, and its proof amounts to the definition of a meta-level function on proofs. However, with proof terms, we can now make that function definition more apparent.

$$\dfrac{A \vdash N_0 : B_k \quad \dfrac{(k \in L)}{B_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\oplus\text{R}'}{A \vdash N_0 \diamond \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\text{CUT}^{B_k} \qquad \oplus_{\ell \in L}\{\ell : B_\ell\} \vdash M : C}{A \vdash (N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M : C}\,\text{A-CUT}^{B}$$

$$=$$

$$\dfrac{A \vdash N_0 : B_k \qquad \dfrac{\dfrac{(k \in L)}{B_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\oplus\text{R}' \qquad \oplus_{\ell \in L}\{\ell : B_\ell\} \vdash M : C}{B_k \vdash \underset{\rightarrow}{k} \blacklozenge M : C}\,\text{A-CUT}^{B}}{A \vdash N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M) : C}\,\text{A-CUT}^{B_k}$$

Figure 8.10: One of the associative cases in the proof of non-analytic cut admissibility (lemma 8.7)

Let $\blacklozenge$ be a nondeterministic binary function on normal terms N and M of compatible types such that $N \blacklozenge M$ is a normal term of the corresponding type:

$$\dfrac{A \vdash N : B \quad B \vdash M : C}{A \vdash N \blacklozenge M : C}\,\text{A-CUT}^{B}\,.$$

Once again, we will prove the cut principle by a lexicographic induction, first on the cut formula, B, and then on the structures of the given terms, N and M. However, because the semi-axiomatic formulation uses different rules than the sequent calculus, the proof's cases are organized a bit differently. In addition to the usual classes of principal, identity, and commutative cases, a new class of associative cases is introduced.

Associative cases Consider, for example, the case $(N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M$. Because the term $\underset{\rightarrow}{k}$ is itself normal, the above term can be reassociated, suggesting that we adopt

$$(N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M = N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)$$

as a clause in the definition of $\blacklozenge$. But is this clause terminating?

Yes, indeed it is. In $N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)$, the inner $\underset{\rightarrow}{k} \blacklozenge M$ terminates because the terms have together become smaller – $\underset{\rightarrow}{k}$ is a proper subterm of $N_0 \diamond \underset{\rightarrow}{k}$ – while the cut formula remains unchanged. The outer $N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)$ also terminates, despite $\underset{\rightarrow}{k} \blacklozenge M$ possibly being larger than M, because the cut formula has become smaller. To aid the reader in tracking the types, fig. 8.10 shows the full typing derivations.

The symmetric case, $N \blacklozenge (\underset{\leftarrow}{k} \diamond M_0)$, is also an associative case and is handled similarly. The complete set of associative clauses is therefore:

$$\begin{aligned}(N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M &= N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)\\ N \blacklozenge (\underset{\leftarrow}{k} \diamond M_0) &= (N \blacklozenge \underset{\leftarrow}{k}) \blacklozenge M_0\,.\end{aligned}$$

Both of these associative cases detach a label and group it together with the neighboring term, thereby enabling interactions between the label and term.

Principal cases Because the above associative cases decompose the analytic cuts $N_0 \diamond \underrightarrow{k}$ and $\underleftarrow{k} \diamond M_0$, the principal cases need only cover those pairings of the $\oplus\text{R}'$ rule with a proof ending in the $\oplus\text{L}$ rule and the symmetric pairings involving the $\&\text{R}$ and $\&\text{L}'$ rules:

$$\underrightarrow{k} \blacklozenge \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$$
$$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge \underleftarrow{k} = N_k$$

If $\underrightarrow{k}$ and $\underleftarrow{k}$ are viewed as directed messages, then these principal clauses in $\blacklozenge$'s definition look much like rules for asynchronous message-passing communication. This observation is at the heart of the Curry–Howard interpretation of singleton logic's semi-axiomatic sequent calculus that we develop in the following chapter.

Identity cases As in the proof of admissibility of cut for the sequent calculus (lemma 8.3), the identity cases cover pairings involving the ID rule and yield the following clauses.

$$\leftrightarrow \blacklozenge M = M$$
$$N \blacklozenge \leftrightarrow = N$$

Commutative cases In the remaining cases, one of the two terms has a top-level constructor that introduces a side formula. For instance, in the cut $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge M$, the constructor $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow -)$ introduces the side formula $\oplus_{\ell \in L}\{\ell : A_\ell\}$. The left-commutative cases yield the following clauses for the definition of $\blacklozenge$.

$$(\underleftarrow{k} \diamond N_0) \blacklozenge M = \underleftarrow{k} \diamond (N_0 \blacklozenge M)$$
$$\underleftarrow{k} \blacklozenge M = \underleftarrow{k} \diamond M$$
$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge M = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell \blacklozenge M)$$

In all of these clauses, the $\blacklozenge$ is permuted with a normal term's top-level constructor. The first clause reassociates cuts, but it has a much different flavor than the above associative cases because $\underleftarrow{k}$ is directed outward here.

There are also several right-commutative cases that are symmetric to the preceding left-commutative cases:

$$N \blacklozenge (M_0 \diamond \underrightarrow{k}) = (N \blacklozenge M_0) \diamond \underrightarrow{k}$$
$$N \blacklozenge \underrightarrow{k} = N \diamond \underrightarrow{k}$$
$$N \blacklozenge \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow M_\ell) = \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N \blacklozenge M_\ell)\,. \qquad \square$$

Notice that the function $\blacklozenge$ defined by this lemma is, in fact, nondeterministic. Many nontrivial critical pairs exist, due to overlapping clauses in the function's definition. For instance, both

$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge \mathsf{caseR}_{k \in K}(k \Rightarrow M_k)$$
$$= \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow \mathsf{caseR}_{k \in K}(k \Rightarrow N_\ell \blacklozenge M_k))$$

and

$$\begin{aligned}&\mathsf{caseL}_{\ell\in L}(\ell \Rightarrow N_\ell) \blacklozenge \mathsf{caseR}_{k\in K}(k \Rightarrow M_k)\\ &\quad = \mathsf{caseR}_{k\in K}(k \Rightarrow \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow N_\ell \blacklozenge M_k))\end{aligned}$$

hold. We conjecture that $\blacklozenge$ is deterministic up to commuting conversions, but will not attempt to prove that result here.

Of course, with the addition of enough side conditions, the function $\blacklozenge$ could be refined into one that is also deterministic at a purely syntactic level. But many of the choices that would be made in breaking ties, such as between the above two terms, seem rather arbitrary, so we prefer to have $\blacklozenge$ remain nondeterministic.

With this lemma in hand, we may finally proceed to proving non-analytic cut elimination.

Theorem 8.8 (Non-analytic cut elimination). *If $A \vdash P : C$, then $A \vdash N : C$ for some normal term N.*

Proof. By structural induction on the proof term P.

This theorem can be phrased as the following admissible rule.

$$\frac{A \vdash P : C}{A \vdash ce(P) : C}\ \textsc{ce}$$

The principal case in proving the admissibility of this rule is:

$$\frac{\dfrac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B}{A \vdash ce(P_1 \diamond P_2) : C}\ \textsc{ce} \quad = \quad \frac{\dfrac{A \vdash P_1 : B}{A \vdash ce(P_1) : C}\ \textsc{ce} \quad \dfrac{B \vdash P_2 : C}{B \vdash ce(P_2) : C}\ \textsc{ce}}{A \vdash ce(P_1) \blacklozenge ce(P_2) : C}\ \textsc{a-cut}^B$$

In this case, a cut is replaced with an appeal to the admissibility of cut. The remaining cases are handled compositionally:

$$\begin{aligned}&ce(P_1 \diamond P_2) = ce(P_1) \blacklozenge ce(P_2)\\ &ce(\leftrightarrow) = \leftrightarrow\\ &ce(\underrightarrow{k}) = \underrightarrow{k}\\ &ce(\mathsf{caseL}_{\ell\in L}(\ell \Rightarrow P_\ell)) = \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow ce(P_\ell))\\ &ce(\mathsf{caseR}_{\ell\in L}(\ell \Rightarrow P_\ell)) = \mathsf{caseR}_{\ell\in L}(\ell \Rightarrow ce(P_\ell))\\ &ce(\underleftarrow{k}) = \underleftarrow{k}\end{aligned}$$

□

8.4 *Extensions of singleton logic*

The usual and semi-axiomatic sequent calculi for singleton logic support various extensions. One simple but useful extension would be to introduce first-order universal and existential quantifiers, $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$, over well-sorted data. Variable typing assumptions, $x{:}\tau$, would not be subject to the single-antecedent restriction – the usual weakening, contraction, and exchange properties apply to variable typing assumptions.

Another direction for extension is to slightly relax the single-antecedent restriction. Instead of demanding that sequents have exactly one antecedent and exactly one consequent, we could allow each sequent to have zero or one antecedents and zero or one consequents. With this relaxation, we arrive at *sub*singleton logic, which now includes the multiplicative units $\mathbf{1}$ and $\bot$.

It is even possible to modify subsingleton logic to include the 'of course' modality, $!A$, from linear logic. To make this work, singleton sequents are extended with a structural zone. Both the right and left rules for $!A$ require that the context be allowed to be empty, so $!A$ is not possible for singleton logic, only subsingleton logic.

Interestingly, the proposition $A \mathbin{!\otimes} B$ with right and left rules

$$\frac{\Gamma; \cdot \vdash B_1 \quad \Gamma; A \vdash B_2}{\Gamma; A \vdash B_1 \mathbin{!\otimes} B_2}\ !\otimes\mathrm{R} \qquad \frac{\Gamma, B_1; B_2 \vdash C}{\Gamma; B_1 \mathbin{!\otimes} B_2 \vdash C}\ !\otimes\mathrm{L}$$

would, strictly speaking, not be possible for singleton logic's sequent calculus: the $!\otimes\mathrm{R}$ rule uses an empty context. But, interestingly, such a proposition would be possible in singleton logic's *semi-axiomatic* sequent calculus: the right rule would be replaced by

$$\frac{}{\Gamma, A_1; A_2 \vdash A_1 \mathbin{!\otimes} A_2}\ !\otimes\mathrm{R},$$

which does not violate the single-antecedent restriction.

8.5 *Other related work*

The singleton sequent calculus (in its infinitary variant) appears independently in work by Fortier and Santocanale on cut elimination for circular proofs of inductive and coinductive types.[16] They seem to have arrived at the single-antecedent restriction from category-theoretic semantic considerations. Fortier and Santocanale do not develop a semi-axiomatic sequent calculus for singleton logic; that is a contribution of this work.

[16] Santocanale 2002; Fortier and Santocanale 2013.

8.5.1 *Connections to Basic Logic*

Sambin et al. (2000) propose a system called Basic Logic in which connectives are defined by a single *definitional equation*. If we apply their ideas to ordered logic, for example, the definitional equation for alternative conjunction would be

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \mathbin{\&} B}\ \&$$

Read top-down, the rule is a *formation* rule; read bottom-up, the rule is two *implicit reflection* rules:

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \mathbin{\&} B}\ \&\mathrm{F} \qquad \frac{\Omega \vdash A \mathbin{\&} B}{\Omega \vdash A}\ \&\mathrm{IR}_1 \qquad \frac{\Omega \vdash A \mathbin{\&} B}{\Omega \vdash B}\ \&\mathrm{IR}_2$$

The formation rule for alternative conjunction is the same as its usual sequent calculus right rule, and the implicit reflection rules correspond to natural deduction elimination rules for $\&$.

To arrive at the usual sequent calculus left rules for alternative conjunction, Sambin et al. proceed by way of what they call *axioms of reflection*. They obtain these axioms by trivializing the implicit reflection rules:

$$\frac{}{A \,\&\, B \vdash A}\ \&\text{A}_1 \qquad \frac{}{A \,\&\, B \vdash B}\ \&\text{A}_2$$

Combining these axioms with CUT, they arrive at *explicit reflection* rules:

$$\frac{\Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ (A \,\&\, B)\ \Omega'_R \vdash C}\ \&\text{ER}_1 \qquad \frac{\Omega'_L\ B\ \Omega'_R \vdash C}{\Omega'_L\ (A \,\&\, B)\ \Omega'_R \vdash C}\ \&\text{ER}_2$$

$$\frac{\dfrac{}{A \,\&\, B \vdash A}\ \&\text{A}_1 \qquad \Omega'_L\ A\ \Omega'_R \vdash C}{\Omega'_L\ (A \,\&\, B)\ \Omega'_R \vdash C}\ \text{CUT}^A$$

Alternatively, the implicit reflection rules could be obtained from the explicit reflection rules by trivializing the explicit rules and then combining the resulting axioms with CUT. So, in fact, the implicit and explicit reflection rules and axioms are all equivalent in the presence of CUT.

$$\frac{\Omega \vdash A \,\&\, B \qquad \dfrac{}{A \,\&\, B \vdash A}\ \&\text{A}_1}{\Omega \vdash A}\ \text{CUT}^{A \& B}$$

These axioms of reflection are exactly the axioms that we use in our semi-axiomatic sequent calculus in place of the usual left rules. (They are also the same as the decomposition rules for $\&$ from the refactored ordered sequent calculus (fig. 5.4) of chapter 5.) Their process of obtaining the explicit reflection rules by combining the axioms of reflection with CUT matches the way that we derive the usual left rules in our semi-axiomatic sequent calculus. However, Sambin et al. appear to have considered the axioms of reflection only as the means to an end, and do not appear to have considered a calculus with these axioms as primitive rules, nor a cut elimination procedure involving the axioms.

9
Semi-axiomatic singleton sequent proofs as session-typed process chains

In the previous chapter, we took a purely proof-theoretic view of singleton logic and its semi-axiomatic sequent calculus. The proof terms assigned to semi-axiomatic sequent proofs were simply syntactic objects, and the proof of non-analytic cut elimination (theorem 8.8) described a meta-level function for normalizing these syntactic objects.

Even in a purely proof-theoretic setting, however, the computational suggestions of these syntactic manipulations were much too strong to ignore: We saw that the principal cases in the proof of admissibility of non-analytic cuts (lemma 8.7) are reminiscent of asynchronous message-passing communication. Following the rich tradition of Curry–Howard isomorphisms between logics and computational systems, this chapter therefore pursues a concurrent computational interpretation of the semi-axiomatic sequent calculus for singleton logic.

$\underset{\sim}{k} \bullet \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$
$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \bullet \underset{\sim}{k} = N_k$

In particular, we will see that singleton propositions can be interpreted as session types that describe patterns of interprocess communication (section 9.1.2); semi-axiomatic sequent proofs can be interpreted as chains of session-typed processes (section 9.1.3); and cut reduction can be interpreted as asynchronous message-passing communication (section 9.1.4). For instance, a proof of $\oplus_{\ell \in L}\{\ell : A_\ell\}$ corresponds to a process that sends a message carrying some label $k \in L$ and then continues communicating according to pattern A_k.

This roughly parallels a recent line of research into a Curry–Howard isomorphism, dubbed Sill, between the intuitionistic linear sequent calculus and session-typed concurrent computation[1] – with two key differences. First, unlike Sill, we use singleton logic, not intuitionistic linear logic. This restricts the process topology to chains,[2] rather than the tree topology that Sill permits. But this restriction should not be seen as a weakness; by leveraging this restriction, computation pipelines can be expressed as a logically motivated fragment.

Second and most importantly, we use the *semi-axiomatic* sequent calculus, rather than a standard sequent calculus like Sill does. The use of semi-

[1] Caires, Pfenning, and Toninho 2016, 2012.

[2] Dezani-Ciancaglini et al. (2014) also consider chains of session-typed processes with unnamed left- and right-hand channels, but not from the logical perspective of a Curry–Howard correspondence.

axiomatic sequent proofs enables a clean and direct interpretation of cut reductions as asynchronous communication, unlike the cut-reductions-as-synchronous-communication view espoused by Sill.[3]

[3] It is possible to give a rather ad hoc treatment of asynchronous communication using Sill's sequent proofs (DeYoung, Caires, et al. 2012), but, in our view, the treatment of asynchronous communication using semi-axiomatic sequent proofs is far more elegant.

We begin, in section 9.1.1, by introducing process chains in their untyped form as finite sequences of processes arranged in a chain topology. Then, in section 9.1.2, we describe the structure of well-typed process expressions that may be used in these chains, and show that the session-typing rules for process expressions correspond to the rules of the semi-axiomatic sequent calculus; section 9.1.3 presents session-typing rules for the process chains themselves. In section 9.1.4, we assign an operational semantics to process chains; this operational semantics arises naturally from the semi-axiomatic proof normalization procedure given in the previous chapter. Lastly, section 9.2 describes coinductively defined type and process definitions.

9.1 *Process chains and process expressions*

9.1.1 *Untyped process chains*

We envision a process chain, C, as a (possibly empty) finite sequence of processes $(P_i)_{i=1}^{n}$, each with its own independent thread of control and arranged in a chain topology. As depicted in the adjacent figure, each process P_i shares unique channels with its left- and right-hand neighbors. Along these channels, neighboring processes may interact and react, changing their own internal state. Because process chains always maintain a chain topology, the channels need not be named – they can instead be referred to as simply the left- and right-hand channels of P_i.

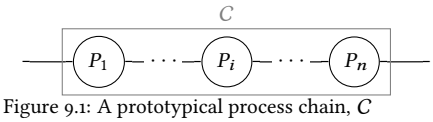


Figure 9.1: A prototypical process chain, C

A chain C does not compute in isolation, however. The left-hand channel of P_1 and the right-hand channel of P_n enable the chain to interact with its surroundings. Because these two channels are the only ones exposed to the external environment, they may be referred to as the left- and right-hand channels of C.

Chains may even be composed end to end by conjoining the right-hand channel of one chain with the left-hand channel of another.

As finite sequences of processes P_i, chains form a free monoid:

$$C, \mathcal{D} ::= (C_1 \parallel C_2) \mid (\cdot) \mid P\,,$$

where $\parallel$ denotes end-to-end composition of chains and $(\cdot)$ denotes the empty chain. As a monoid, chains are subject to associativity and unit laws (see adjacent figure). We do not distinguish chains that are equivalent up to these laws, instead treating such chains as syntactically identical.

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$(\cdot) \parallel C = C = C \parallel (\cdot)$$

Figure 9.2: Monoid laws for process chains

The notation for a composition $C_1 \parallel C_2$ is intended to recall parallel composition of π-calculus processes, $P_1 \mid P_2$. However, unlike π-calculus composition, parallel composition of chains is *not* commutative because the sequential order of processes within a chain matters.

Table 9.1: Singleton session types

Judgmental rules	
$P_1 \diamond P_2$	Spawn new, neighboring threads of control for P_1 and P_2, then terminate the current thread of control
$\leftrightarrow$	Terminate the current thread of control
Internal choice, $\oplus_{\ell \in L}\{\ell : A_\ell\}$	
$\underrightarrow{k}$, with $k \in L$	A message to the right-hand client, carrying label k
$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$	Await a message $\underrightarrow{k}$ from the left-hand provider, then continue as P_k
External choice, $\&_{\ell \in L}\{\ell : A_\ell\}$	
$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$	Await a message $\underleftarrow{k}$ from the right-hand client, then continue as P_k
$\underleftarrow{k}$, with $k \in L$	A message to the left-hand provider, carrying label k

9.1.2 *Session-typed process expressions*

Thus far, we have examined the overall structure of (untyped) process chains without detailing the internal structure of individual processes. We now turn to the specifics of well-typed processes.

As previously alluded, each of a chain's processes constitutes its own, independent thread of control dedicated to executing the instructions described by a process expression P. Processes are thus dynamic realizations of the static process expressions, in the same way that executables run source code.[4]

[4] It is occasionally convenient to blur this distinction, so we sometimes abuse terminology and refer to a process expression P as a "process".

Reinterpreting the proof-term judgment of singleton logic's semi-axiomatic sequent calculus, we arrive at a session-typing judgment for process expressions. The judgment

$$A \vdash P : B$$

now means that P is the expression for a process that offers, along its right-hand channel, the service described by the session type B, while concurrently using, along its left-hand channel, the service described by the type A. In other words, the right-hand neighbor acts as a client of service B from P, while the left-hand neighbor of process P acts as a provider of service A to P.

Session types describe the patterns by which a process is permitted to communicate with its left- and right-hand neighbors.

Under this reinterpretation of the basic judgment, the proof rules of the singleton Hilbert system become session-typing rules for process expressions. Specifically, the right rules define what it means for a provider to offer a particular service, while the left rules show how a client may use that service.

As an example, consider additive disjunction and its proof rules. From a computational perspective, an additive disjunction $\oplus_{\ell \in L}\{\ell : A_\ell\}$ is inter-

preted as an internal choice, the type of a process that sends some label $k \in L$ to its right-hand client and then behaves like A_k. Recall the $\oplus\text{R}'$ and $\oplus\text{L}$ rules:

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}} \oplus\text{R}' \qquad \frac{\forall \ell \in L\colon \quad A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C} \oplus\text{L}$$

The proof term $\underrightarrow{k}$ is now reinterpreted as the expression for a message, sent to the right-hand client (as the arrow suggests), that carries the label k as its payload. The proof term $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is reinterpreted as the expression for a client process that awaits a message $\underrightarrow{k}$ from its left-hand provider and then continues the thread of control with the corresponding branch, P_k.

Additive conjunction, $\&_{\ell \in L}\{\ell : A_\ell\}$, is interpreted dually as external choice, the type of a process that awaits a label $k \in L$ from its client and then behaves like A_k.

$$\frac{\forall \ell \in L\colon \quad A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}} \&\text{R} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k} \&\text{L}'$$

As might be expected, the proof terms $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$ and $\underleftarrow{k}$ are interpreted symmetrically to internal choice's $\underrightarrow{k}$ and $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ expressions: $\underleftarrow{k}$ is a message to the left-hand provider, and $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is an input process that awaits a message from the right-hand client.

The proof term $P_1 \diamond P_2$ for composition of proofs is now reinterpreted as the expression for a process that will spawn new, neighboring threads of control for P_1 and P_2 and then terminate the original thread of control. In effect, $P_1 \diamond P_2$ now composes process expressions.

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \text{CUT}^B$$

For $P_1 \diamond P_2$ to be a well-typed composition, P_1 must offer the same service that P_2 uses.

Proof-theoretically, the identity and cut rules are inverses, so we should expect their process interpretations to be similarly inverse. The process expression $P_1 \diamond P_2$ spawns threads of control, so $\leftrightarrow$, as its inverse, terminates the thread of control.

$$\frac{}{A \vdash \leftrightarrow : A}\ \text{ID}^A$$

9.1.3 *Session-typed process chains*

With the session-typing system for process expressions in hand, session types can be assigned to process chains, too. We use a session-typing judgment

$$A \Vdash C : B,$$

meaning that the chain C offers, along its right-hand channel, the service B, while concurrently using, along its left-hand channel, the service A. Similar to

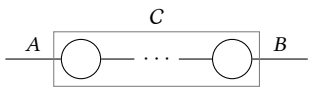


Figure 9.3: A well-typed process chain that uses service A to offer service B

individual processes, a chain C thus enjoys client and provider relationships with its left- and right-hand environments, respectively.

The simplest session-typing rule for chains is the one that types a chain consisting of a single running process P:

$$\frac{A \vdash P : B}{A \Vdash P : B}\ \textsc{c-proc}$$

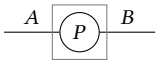


Figure 9.4: A chain made of one well-typed process that uses service A to offer service B

In other words, a running process has the same type as its underlying process expression.

The session-typing rule for the empty chain, $(\cdot)$, is also fairly direct. The empty chain offers a service A to its right-hand client by using the service of its left-hand provider:

$$\frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A$$

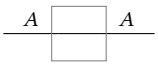


Figure 9.5: A well-typed empty chain that uses service A to offer service A

Save for the contrasting $\Vdash$ turnstile and the empty chain in place of a forwarding process expression, this mirrors the $\textsc{id}^A$, the identity rule for process expressions. We will see shortly that this is not a coincidence.

Finally, a parallel composition of chains, $C_1 \parallel C_2$, is well-typed only if C_1 offers the same service that C_2 uses, otherwise communication between C_1 and C_2 would be mismatched. This condition is reflected in a cut principle for the session-typing judgment:

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B$$

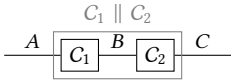


Figure 9.6: A well-typed process chain that uses service A to offer service B

Once again, there are strong similarities to a process expression – $P_1 \diamond P_2$ and its $\textsc{cut}^B$ session-typing rule, in this case. We can make these similarities explicit by defining a homomorphism, $(-)^\diamond$, from chains to process expressions: This homomorphism is type-preserving:

$$(C_1 \parallel C_2)^\diamond = C_1^\diamond \diamond C_2^\diamond$$
$$(\cdot)^\diamond = \leftrightarrow$$
$$P^\diamond = P$$

Figure 9.7: A homomorphism from chains to process expressions

Theorem 9.1. *If $A \Vdash C : B$, then $A \vdash C^\diamond : B$.*

Proof. By structural induction on the session-typing derivation. □

At first, the distinction between offering and using a service may seem a bit odd, given that we placed so much emphasis on the symmetry of singleton sequents $A \vdash B$. Singleton sequents are indeed symmetric, as theorem 8.2 showed. But imposing a provider–client, offer–use distinction is useful for placing our process chains and expressions within existing conceptual frameworks for session-typed concurrency. In particular, the distinction helps to relate this process interpretation of singleton logic back to the Sill interpretation of linear logic.[5]

[5] Caires and Pfenning 2010; Caires, Pfenning, and Toninho 2012, 2016; Toninho et al. 2013.

9.1.4 *From admissibility of non-analytic cuts to an operational semantics*

In the previous chapter, we presented a procedure for normalizing semi-axiomatic sequent proofs in singleton logic. Proof normalization was important

Figure 9.8: Well-typed process expressions and process chains

$$\textsc{session types} \quad A, B, C ::= \alpha \mid \oplus_{\ell \in L}\{\ell : A_\ell\} \mid \&_{\ell \in L}\{\ell : A_\ell\}$$

$$\textsc{process chains} \quad \mathcal{C}, \mathcal{D} ::= (\mathcal{C}_1 \parallel \mathcal{C}_2) \mid \cdot \mid P$$

$$(\mathcal{C}_1 \parallel \mathcal{C}_2) \parallel \mathcal{C}_3 = \mathcal{C}_1 \parallel (\mathcal{C}_2 \parallel \mathcal{C}_3)$$
$$(\cdot) \parallel \mathcal{C} = \mathcal{C} = \mathcal{C} \parallel (\cdot)$$

$$\frac{A \Vdash \mathcal{C}_1 : B \quad B \Vdash \mathcal{C}_2 : C}{A \Vdash \mathcal{C}_1 \parallel \mathcal{C}_2 : C}\ \textsc{c-cut}^B \qquad \frac{}{A \vdash \cdot : A}\ \textsc{c-id}^A \qquad \frac{A \vdash P : B}{A \Vdash P : B}\ \textsc{c-proc}$$

$$\textsc{process expressions} \quad P, Q ::= P_1 \diamond P_2 \mid \leftrightarrow \mid \underrightarrow{k} \mid \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \underleftarrow{k}$$

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \leftrightarrow : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon \quad A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon \quad A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\textsc{l}'$$

to establish that the singleton logic has a well-defined meaning-theoretic explanation.

In this chapter, however, our perspective has shifted from proof theory to concurrent computation, from proofs to processes. And so normalization is no longer appropriate – we now want to expose the concurrent computational behavior, not just normal forms. The situation is analogous to that of intuitionistic natural deduction and simply-typed functional computation: there, proof normalization occurs in the premise of the implication introduction rule but the usual operational semantics for functional computation does not reduce under function abstractions.

In fact, the difference is even starker here because, once recursive process definitions are introduced (section 9.2), many useful processes will be nonterminating. Thus, there is no clear notion of value, as exists in functional computation. Nevertheless, in good Curry–Howard fashion, the principal cases of semi-axiomatic proof normalization will still directly inform the operational semantics of processes.

In the previous section, the description of how proof terms are reinterpreted as process expressions already hinted at a computational strategy. Here we present that operational semantics in its full detail.

The operational semantics for process chains is centered around *reduction*, a binary relation on chains which we write as $\longrightarrow$; we will use $\Longrightarrow$ for the reflexive, transitive closure of reduction. Reductions may occur among any of the chain's processes, and thus the relation is compatible with the monoid operation, $\|$:

$$\frac{C_1 \longrightarrow C_1'}{C_1 \parallel C_2 \longrightarrow C_1' \parallel C_2} \qquad \frac{C_2 \longrightarrow C_2'}{C_1 \parallel C_2 \longrightarrow C_1 \parallel C_2'}$$

At the heart of reduction are two symmetric rules that describe how messages are received:

$$\frac{(k \in L)}{\underrightarrow{k} \parallel \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \longrightarrow P_k} \qquad \frac{(k \in L)}{\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel \underleftarrow{k} \longrightarrow P_k}$$

As suggested earlier, when a process $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ receives a message from its left-hand provider, it continues the thread of control with the indicated branch, P_k; the rule involving $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is symmetric. These two reduction rules mimic the principal proof normalization steps for singleton logic's semi-axiomatic sequent proofs: $\underrightarrow{k} \blacklozenge \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$ and $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge \underleftarrow{k} = N_k$.

As suggested earlier, a process $P_1 \diamond P_2$ spawns, in place, new neighboring threads of control for P_1 and P_2, respectively, while the original thread of control terminates; and a process $\leftrightarrow$ terminates its thread of control. The operational semantics formalizes these notions in rules that decompose $P_1 \diamond P_2$

and $\leftrightarrow$:

$$\frac{}{P_1 \diamond P_2 \longrightarrow P_1 \parallel P_2} \qquad \frac{}{\leftrightarrow \longrightarrow (\cdot)}$$

Because process chains are always considered up to associativity and unit laws, these reduction rules (along with the above $\parallel$-compatibility rules) reflect the associative and identity normalization steps in the proof of admissibility of non-analytic cuts (lemma 8.7). For example, just as

$$(N_0 \diamond \underrightarrow{k}) \bullet M = N_0 \bullet (\underrightarrow{k} \bullet M)$$

is an associative normalization step,

$$(P_0 \diamond \underrightarrow{k}) \parallel P_1 \longrightarrow\!\!= P_0 \parallel (\underrightarrow{k} \parallel P_1)$$

is a reduction. Similarly, $\leftrightarrow \parallel P \longrightarrow\!\!= P$ is a reduction that reflects the normalization step $\leftrightarrow \bullet M = M$.

These rules witness the close connection between proof normalization and the operational semantics of processes, but one class of normalization steps does not have a direct analogue in the operational semantics: the class of commutative normalization steps. As a prototypical example, recall the step involving $\mathsf{caseL}()$:

$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \bullet M = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell \bullet M)\,.$$

As part of proof normalization, this step is quite natural because it progresses toward a normal form by pushing the admissible cut, represented by the $\bullet$ constructor, further in and pulling the $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow -)$ construction out. In the operational semantics, however, it would be wrong to have the corresponding

$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel C \longrightarrow \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell \parallel C)$$

as a reduction – it inappropriately mixes dynamic and static objects by bringing the chain C within the process expression $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow -)$.

The session-typing rules and operational semantics enjoy preservation and progress theorems.

Theorem 9.2 (Preservation). *If $A \Vdash C : B$ and $C \longrightarrow C'$, then $A \Vdash C' : B$.*

Proof. By structural induction on the given reduction, $C \longrightarrow C'$. □

Theorem 9.3 (Progress). *If $A \Vdash C : B$, then either:*

- *chain C can reduce, that is, $C \longrightarrow C'$;*
- *chain C is waiting to interact with its right-hand client, that is, $C = C' \parallel \underrightarrow{k}$ or $C = C' \parallel \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$;*
- *chain C is waiting to interact with its left-hand provider, that is, $C = \underleftarrow{k} \parallel C'$ or $C = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel C'$; or*
- *chain C is empty, that is, $C = (\cdot)$.*

Proof. By structural induction on the typing derivation, $A \Vdash C : B$. □

9.2 *Coinductively defined types and process expressions*

Unfortunately, there are many relatively simple computational behaviors that cannot be described by the finitary session types thus far. For instance, a transducer process that receives, one-by-one, a stream of input symbols and forms an output stream by replacing each b with an a cannot be represented.

The solution is to introduce coinductively defined types, in a manner reminiscent of the coinductively defined propositions, $\hat{p}^- \triangleq A^-$, seen in section 6.1.3. We will often write coinductively defined types with Greek letters, such as $\alpha \triangleq A$. Coinductively defined types are not particularly useful if process expressions remain finitary, so we also introduce mutually coinductively defined process expressions: $A \vdash \hat{p} : C \triangleq P$.

That these definitions are coinductive is guaranteed by requiring that along every cycle among defined types and processes there is a type constructor or process expression constructor.[6] This justifies an *equi*recursive treatment of types in which type definitions may be silently unfolded (or re-folded) at will.

[6] This generalizes the local contractivity condition described by Gay and Hole (2005).

These mutually coinductive definitions are collected into signatures Ξ. To verify that the definitions are well-formed, we use a judgment $\vdash_{\Xi'} \Xi$, where the definitions Ξ are judged according to the definitions Ξ'. The mutually coinductive knot can be tied by using $\vdash_{\Xi} \Xi$ at the top level: Ξ is well-formed if $\vdash_{\Xi} \Xi$ holds.

$$\frac{}{\vdash_{\Xi'} (\cdot)} \qquad \frac{\vdash_{\Xi'} \Xi \quad A \vdash_{\Xi'} P : C}{\vdash_{\Xi'} \Xi, (A \vdash \hat{p} : C \triangleq P)}$$

All of the process expression typing judgments are indexed by a well-formed signature Ξ. A call to $\hat{p}$ has type $A \vdash C$ if $A \vdash \hat{p} : C\hat{P}$ is listed in Ξ:

$$\frac{(A \vdash \hat{p} : C \triangleq P) \in \Xi}{A \vdash_{\Xi} \hat{p} : C}$$

As with coinductively defined propositions, these type and process expression definitions are coinductive in only a syntactic sense. In particular, the coinductively defined process expressions are not necessarily behaviorally coinductive, *i.e.*, productive. For example, $\hat{p} \triangleq \mathsf{caseL}(a \Rightarrow \underset{\rightarrow}{a} \diamond p)$ is not a productive process – after receiving an initial message $\underset{\rightarrow}{a}$, p diverges without sending or receiving any further messages.

Once coinductively defined types and process expressions are added, there is, strictly speaking, no longer a Curry–Howard isomorphism between session-typed process chains and the semi-axiomatic proofs of singleton logic. Importantly, however, the core system remains unchanged and still enjoys the isomorphism because the unbounded behavior is added modularly. The situation is once again analogous to the Curry–Howard isomorphism between intuitionistic natural deduction and the simply-typed λ-calculus: When the λ-calculus is extended with recursive types and functions, the meaning of the type constructor for simple function types remains unchanged and still isomorphic with intuitionistic implication.

Just as a Curry–Howard isomorphism can be recovered in the λ-calculus when general recursive types are restricted to inductive and coinductive types, work by Derakhshan and Pfenning[7] and Somayyajula and Pfenning[8] shows that a Curry–Howard isomorphism between session-typed process chains and (sub-)singleton sequent calculus proofs can be recovered if types are inductive or behaviorally coinductive.

[7] Derakhshan and Pfenning 2020.
[8] Somayyajula and Pfenning 2020.

9.3 *Examples*

9.3.1 *Binary counter*

We can once again revisit binary counters, this time as an example of session-typed process chains.

SESSION TYPES A natural number in binary will be represented by a process chain of type $\epsilon \Vdash \kappa$, where κ is a session type that describes the service that a counter offers and ϵ is a type parameter that represents a terminated counter.

The type κ of counters is given by

$$\kappa \triangleq \&\{i : \kappa, d : \oplus\{z : \epsilon, s : \kappa\}\}$$

and describes a service that offers the client a choice between incrementing (i branch) and decrementing (d branch) the counter:

- If the client chooses to increment the counter, then the incremented counter again offers the same service, κ, to the client.
- Otherwise, if the client chooses to decrement the counter, then the provider replies with either $\underrightarrow{z}$ or $\underrightarrow{s}$ depending on the counter's value.
  - If the counter's value is 0, it cannot be decremented further and so it emits $\underrightarrow{z}$ and then terminates at type ϵ.
  - If the counter's value is some strictly positive natural number n, then the provider signals that by emitting $\underrightarrow{s}$ and then continues as a counter of value $n - 1$ that offers service κ.

As a shorthand, we can introduce a type ρ of decrement responses:

$$\begin{aligned} \kappa &\triangleq \&\{i : \kappa, d : \rho\} \\ \rho &\triangleq \oplus\{z : \epsilon, s : \kappa\}\,. \end{aligned}$$

PROCESS EXPRESSIONS A counter will be a big-endian chain of processes $\hat{b}_0$ and $\hat{b}_1$, prefixed by a process $\hat{e}$. For example, the chain $\hat{e} \parallel \hat{b}_1 \parallel \hat{b}_0$ would represent a counter of value 2. For these chains to have type $\epsilon \Vdash \kappa$, the process expressions $\hat{b}_0$ and $\hat{b}_1$ must have type $\kappa \vdash \kappa$, while $\hat{e}$ must have type $\epsilon \vdash \kappa$. In addition, to implement decrements involving $\hat{b}_0$, it is also convenient to have a coinductively defined process expression $\hat{b}'_0$ of type $\rho \vdash \rho$.

These process expressions are defined by

$$\epsilon \vdash \hat{e} : \kappa \triangleq \mathsf{caseR}(i \Rightarrow \hat{e} \diamond \hat{b}_1 \mid d \Rightarrow \underline{z})$$
$$\kappa \vdash \hat{b}_0 : \kappa \triangleq \mathsf{caseR}(i \Rightarrow \hat{b}_1 \mid d \Rightarrow \underline{d} \diamond \hat{b}'_0)$$
$$\kappa \vdash \hat{b}_1 : \kappa \triangleq \mathsf{caseR}(i \Rightarrow \underline{i} \diamond \hat{b}_0 \mid d \Rightarrow \hat{b}_0 \diamond \underline{s})$$
$$\rho \vdash \hat{b}'_0 : \rho \triangleq \mathsf{caseL}(z \Rightarrow \underline{z} \mid s \Rightarrow \hat{b}_1 \diamond \underline{s})$$

For instance, a $\hat{b}_1$ process that receives the $\underline{i}$ increment message will spawn, in place, neighboring threads of control for $\underline{i}$ and $\hat{b}_0$ and then terminate the original thread of control. In effect, this sends the $\underline{i}$ increment message to the more significant bits as a carry and flips this bit to $\hat{b}_1$.

As a second example, a $\hat{b}_0$ process that receives the $\underline{d}$ decrement message should decrement the counter formed by the more significant bits and then analyze the response with a $\hat{b}'_0$ process.

- If that $\hat{b}'_0$ process receives a response of $\underline{z}$, then the more significant bits had value 0 and so must the counter as a whole. According to the type ρ, the current thread of control must produce a response of type $\epsilon \vdash \rho$, which is easily done by sending $\underline{z}$.

- Otherwise, if the $\hat{b}'_0$ process receives a response of $\underline{s}$, then the more significant bits had value $n+1$, for some $n \geq 0$, and the counter as a whole must have value $2n+1$ after the decrement. This is accomplished by emitting $\underline{s}$ and replacing $\hat{b}'_0$ with a recursive call to $\hat{b}_1$.

As an example of these processes in action, observe that $\hat{e} \parallel \hat{b}_1 \parallel \underline{i} \parallel \underline{d}$ has the following trace, among others.

$$\begin{aligned}
&\hat{e} \parallel \hat{b}_1 \parallel \underline{i} \parallel \underline{d} \\
&\quad\longrightarrow \hat{e} \parallel (\underline{i} \diamond \hat{b}_0) \parallel \underline{d} \longrightarrow \hat{e} \parallel \underline{i} \parallel \hat{b}_0 \parallel \underline{d} \\
&\quad\longrightarrow \hat{e} \parallel \underline{i} \parallel (\underline{d} \diamond \hat{b}'_0) \longrightarrow \hat{e} \parallel \underline{i} \parallel \underline{d} \parallel \hat{b}'_0 \\
&\quad\longrightarrow (\hat{e} \diamond \hat{b}_1) \parallel \underline{d} \parallel \hat{b}'_0 \longrightarrow \hat{e} \parallel \hat{b}_1 \parallel \underline{d} \parallel \hat{b}'_0 \\
&\quad\longrightarrow \hat{e} \parallel (\hat{b}_0 \diamond \underline{s}) \parallel \hat{b}'_0 \longrightarrow \hat{e} \parallel \hat{b}_0 \parallel \underline{s} \parallel \hat{b}'_0 \\
&\quad\longrightarrow \hat{e} \parallel \hat{b}_0 \parallel (\hat{b}_1 \diamond \underline{s}) \longrightarrow \hat{e} \parallel \hat{b}_0 \parallel \hat{b}_1 \parallel \underline{s}
\end{aligned}$$

Alternatively, we can give process definitions for i and d and treat e, b_0, and b_1 as messages. This is the analogue of the functional choreography from section 6.4.2, as we will discuss more in chapter 10.

$$\hat{\imath} \triangleq \mathsf{caseL}(e \Rightarrow \underline{e} \diamond \underline{b}_1 \mid b_0 \Rightarrow \underline{b}_1 \mid b_1 \Rightarrow \hat{\imath} \diamond \underline{b}_0)$$
$$\hat{d} \triangleq \mathsf{caseL}(e \Rightarrow \underline{z} \mid b_0 \Rightarrow \hat{d} \diamond \hat{b}'_0 \mid b_1 \Rightarrow \underline{b}_0 \diamond \underline{s})$$
$$\hat{b}'_0 \triangleq \mathsf{caseL}(z \Rightarrow \underline{z} \mid s \Rightarrow \hat{d} \diamond \hat{b}'_0)$$

9.3.2 *Sequential transducers*

By this point in this document, the reader will likely expect either DFAs or NFAs as our next example of session-typed process chains. Instead, we will use sequential transducers, as introduced in chapter 2.

SESSION TYPES We should first construct a type that describes the words over an alphabet Σ, respectively. However, because the language formulated in this chapter does not have inductive types, we cannot describe the finite words Σ^* alone. With the inductive session types studied by Derakhshan and Pfenning[9] and Somayyajula and Pfenning,[10] that would be possible, but we do not pursue that extension here.

[9] Derakhshan and Pfenning 2020.
[10] Somayyajula and Pfenning 2020.

Instead, we will construct a type that describes the *infinite* words over alphabet Σ:

$$\Sigma^\omega \triangleq \oplus_{a\in\Sigma}\{a : \Sigma^\omega\}\,.$$

A process that offers type Σ^ω is one that emits a sequence of messages that correspond to an infinite word over Σ. For instance, when given by the following definition, $\epsilon \vdash \hat{w} : \Sigma^\omega$ is a process that corresponds to the infinite word $w = abbabb\cdots$:

$$\epsilon \vdash \hat{w} : \Sigma^\omega \triangleq \hat{w} \diamond (\underset{\rightarrow}{b} \diamond \underset{\rightarrow}{b} \diamond \underset{\rightarrow}{a})\,.$$

Notice that the type parameter ϵ is never used directly and could be replaced with any type A.

Using this type, we can now implement *infinite*-word sequential transducers with well-typed process expressions.

PROCESS EXPRESSIONS Let $\mathcal{T} = (Q, \delta, \sigma)$ be an infinite-word sequential transducer over the input and output alphabets Σ and Γ, respectively. Each state $q \in Q$ maps words from Σ^ω to Γ^ω, and therefore ought to correspond to a process expression $\hat{q}$ of type $\Sigma^\omega \vdash \Gamma^\omega$:

$$\Sigma^\omega \vdash \hat{q} : \Gamma^\omega \triangleq \mathsf{case}\mathsf{L}_{a\in\Sigma}(a \Rightarrow \hat{q}' \diamond \underset{\rightarrow}{w}^{\mathrm{R}}) \quad \text{where } \delta(q,a) = q' \text{ and } \sigma(q,a) = w.$$

(The anti-homomorphism $\underset{\rightarrow}{w}^{\mathrm{R}}$ is defined in the adjacent figure. It is notationally identical to, but formally distinct from, the anti-homomorphism from words to contexts of right-directed atoms (fig. 6.7). However, as we will see in chapter 10, the two anti-homomorphisms are conceptually related.)

$$\underset{\rightarrow}{w}^{\mathrm{R}} = \begin{cases} \leftrightarrow & \text{if } w = \epsilon \\ \underset{\rightarrow}{w_0}^{\mathrm{R}} \diamond \underset{\rightarrow}{a} & \text{if } w = a\,w_0 \end{cases}$$

Figure 9.9: An anti-homomorphism from Γ^* to processes of type $\Gamma^\omega \vdash \Gamma^\omega$

For a concrete example, recall from chapter 2 the infinite-word sequential transducer over $\Sigma = \Gamma = \{a, b\}$ (repeated in the adjacent figure) that compresses each run of bs within the input word into a single b. Because $\Sigma = \Gamma = \{a, b\}$, the types for input and output words are defined as:

$$\Sigma^\omega \triangleq \oplus\{a : \Sigma^\omega, b : \Sigma^\omega\}$$
$$\Gamma^\omega \triangleq \oplus\{a : \Gamma^\omega, b : \Gamma^\omega\}\,.$$

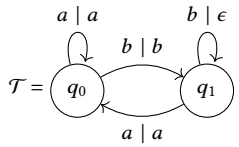


Figure 9.10: An infinite-word sequential transducer that compresses runs of consecutive bs. (Repeated from fig. 2.4.)

Following the encoding laid out above, the states q_0 and q_1 become process expressions $\hat{q}_0$ and $\hat{q}_1$ of type $\Sigma^\omega \vdash \Gamma^\omega$ defined by:

$$\Sigma^\omega \vdash \hat{q}_0 : \Gamma^\omega \triangleq \mathsf{caseL}(a \Rightarrow \hat{q}_0 \diamond \underrightarrow{a} \mid b \Rightarrow \hat{q}_1 \diamond \underrightarrow{b})$$
$$\Sigma^\omega \vdash \hat{q}_1 : \Gamma^\omega \triangleq \mathsf{caseL}(a \Rightarrow \hat{q}_0 \diamond \underrightarrow{a} \mid b \Rightarrow \hat{q}_1 \diamond \leftrightarrow) .$$

Sequential transducers are closed under composition. To represent the composition of two infinite-word sequential transducers $\mathcal{T}_1$ and $\mathcal{T}_2$ as a well-typed process, we could simply construct their composition, $\mathcal{T} = \mathcal{T}_2 \circ \mathcal{T}_1$, as a sequential transducer in its own right and then represent the transducer $\mathcal{T}$ as a well-typed process.

Even easier, however, is to directly compose the processes that represent the transducers $\mathcal{T}_1$ and $\mathcal{T}_2$. If $\mathcal{T}_1$ and $\mathcal{T}_2$ are in states q and s, respectively, then the processes $\Sigma^\omega \vdash \hat{q} : \Gamma^\omega$ and $\Gamma^\omega \vdash \hat{s} : \Delta^\omega$ are well-typed and the process $\Sigma^\omega \vdash \hat{q} \diamond \hat{s} : \Delta^\omega$ describes the current state of the composition, $\mathcal{T} = \mathcal{T}_2 \circ \mathcal{T}_1$. In fact, DeYoung and Pfenning[11] prove – in a very closely related, if slightly different, framework – that cut elimination actually constructs a normal-form process for the transducer $\mathcal{T}$.

[11] DeYoung and Pfenning 2016.

9.3.3 *Turing machines*

TWO-WAY INFINITE TAPE TURING MACHINE Let $\mathcal{M} = (Q, \delta)$ be a two-way infinite tape Turing machine over alphabet Σ. We imagine the two-way infinite tape as divided into two one-way infinite halves with the machine's finite control, or head, sitting between them. Each of these halves will be represented as a stream of symbols from Σ, directed inward toward $\mathcal{M}$'s head. As such, these two one-way infinite halves are described by the following dual types:[12]

$$\Sigma^\omega \triangleq \oplus_{a \in \Sigma}\{a : \Sigma^\omega\}$$
$$(\Sigma^\omega)^\perp \triangleq \&_{a \in \Sigma}\{a : (\Sigma^\omega)^\perp\} .$$

[12] The involution $(-)^\perp$ was defined in fig. 8.5, but not for coinductively defined propositions. Here we use $(\Sigma^\omega)^\perp$ merely as the name of a coinductively defined type, though the choice of name is intended to evoke the duality with the type Σ^ω.

That is, a process of type $A \vdash \Sigma^\omega$, for some A, acts as the left-hand one-way infinite half of the tape, whereas a process of the dual type, $(\Sigma^\omega)^\perp \vdash B$ for some B, acts as the right-hand one-way infinite half of the tape.

Because the machine's head sits between these two halves of the two-way infinite tape, it ought to correspond to a process of type $\Sigma^\omega \vdash (\Sigma^\omega)^\perp$. Indeed, for each state $q \in Q$, we will define two process expressions, $\Sigma^\omega \vdash {}^\triangleleft\hat{q} : (\Sigma^\omega)^\perp$ and $\Sigma^\omega \vdash \hat{q}^\triangleright : (\Sigma^\omega)^\perp$, for the two heads possible in state q:

$$\Sigma^\omega \vdash {}^\triangleleft\hat{q} : (\Sigma^\omega)^\perp \triangleq \mathsf{caseL}_{a \in \Sigma}\left(a \Rightarrow \begin{cases} {}^\triangleleft\hat{q}' \diamond \underleftarrow{b} & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ \underrightarrow{b} \diamond \hat{q}'^\triangleright & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases}\right)$$
$$\Sigma^\omega \vdash \hat{q}^\triangleright : (\Sigma^\omega)^\perp \triangleq \mathsf{caseR}_{a \in \Sigma}\left(a \Rightarrow \begin{cases} {}^\triangleleft\hat{q}' \diamond \underleftarrow{b} & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ \underrightarrow{b} \diamond \hat{q}'^\triangleright & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases}\right)$$

That is, a left-facing head begins by reading a symbol from its left. Depending on the symbol that is read, the machine writes a new symbol in its place and

then either advances the head to the left or otherwise turns the head to face the right. Writing a new symbol is accomplished by spawning a message directed toward the head. Right-facing heads are symmetric.

Notice that the tape in this process implementation is truly two-way infinite. This is consistent with the model presented in chapter 2, but differs from the traditional model of a Turing machine. In the traditional model, the tape might more accurately be described as two-way *unbounded* – at any given moment, the tape is finite, but can be extended. If we wanted to prove the adequacy of our process implementation adequate with respect to traditional Turing machines, we would have to prove adequacy for infinite tapes that behave as merely unbounded, using a distinguished blank symbol.

ONE-WAY INFINITE TAPE TURING MACHINE Because the head of a two-way infinite tape machine corresponds to a process of type $\Sigma^\omega \vdash (\Sigma^\omega)^\bot$, it is not possible to easily compose two such machines using cut, as we had done for sequential transducers. The types simply do not match:

$$\Sigma^\omega \vdash {}^{\triangleleft}\hat{q} : (\Sigma^\omega)^\bot \neq \Sigma^\omega \vdash \hat{s}^{\triangleright} : (\Sigma^\omega)^\bot .$$

However, if we use one-way infinite tape Turing machines, composition of machines is possible.

We will use the following types. The type Σ^ω is the same as above, but we introduce a type $(\Sigma^*)^\bot$ that will be used in describing the one-way infinite tape's finite end.

$$\Sigma^\omega \triangleq \oplus_{a\in\Sigma}\{a : \Sigma^\omega\}$$
$$(\Sigma^*)^\bot \triangleq \&_{a\in\Sigma}\{a : (\Sigma^*)^\bot, \$: \Sigma^\omega\}$$

In particular, the label \$ will be used to mark the tape's finite end.

The type name $(\Sigma^*)^\bot$ was chosen to suggest a left-directed finite string, but should not be taken too literally – as previously mentioned, the language presented in this chapter does not have the inductive types that would be necessary to enforce finiteness. Notice, too, that any process that offers the type $(\Sigma^*)^\bot$ continues by offering Σ^ω when $\underset{\leftarrow}{\$}$ is received, which will be crucial in composing machines (although not exactly what would be expected of a strict dual of Σ^*).

The machine's head ought to correspond to a process of type $\Sigma^\omega \vdash (\Sigma^*)^\bot$. For each state $q \in Q$, we will define a process $\Sigma^\omega \vdash {}^{\triangleleft}\hat{q} : (\Sigma^*)^\bot$ just like we did for the two-way infinite tape machines:

$$\Sigma^\omega \vdash {}^{\triangleleft}\hat{q} : (\Sigma^*)^\bot \triangleq \mathsf{case}\mathsf{L}_{a\in\Sigma}\left(a \Rightarrow \begin{cases} {}^{\triangleleft}\hat{q}' \diamond \underset{\leftarrow}{b} & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ \underset{\rightarrow}{b} \diamond \hat{q}'^{\triangleright} & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases}\right)$$

For each state $q \in Q$, we will also define a process $\Sigma^\omega \vdash \hat{q}^{\triangleright} : (\Sigma^*)^\bot$ for the right-facing head in state q. This process is mainly like the process definition for a right-facing head in the two-way infinite tape Turing machine. The

difference is that here we have a \$ branch, owing to the presence of label \$ in the type $(\Sigma^*)^\bot$:

$$\Sigma^\omega \vdash \hat{q}^{\triangleright} : (\Sigma^*)^\bot \triangleq \mathsf{case}\mathsf{R}_{a\in\Sigma}\left(\begin{array}{ll} a \Rightarrow & \begin{cases} {}^{\triangleleft}\hat{q}' \diamond \underset{\leftarrow}{b} & \text{if } \delta(q,a) = (q', b, \mathsf{L}) \\ \underset{\rightarrow}{b} \diamond \hat{q}'^{\triangleright} & \text{if } \delta(q,a) = (q', b, \mathsf{R}) \end{cases} \\ \mid \$ \Rightarrow & \begin{cases} {}^{\triangleleft\!\triangleright} & \text{if } q \in F \\ {}^{\triangleleft}\hat{q} \diamond \underset{\leftarrow}{\$} & \text{otherwise} \end{cases} \end{array}\right)$$

Just like the mathematical model of a one-way infinite tape Turing machine, the behavior of a right-facing head that reaches the finite end of the tape depends on whether the current state is final. If it is final, then the forwarding process causes the machine's head to terminate, leaving the tape. Otherwise, if the state is not final, then the head is turned to the left using ${}^{\triangleleft}\hat{q} \diamond \underset{\leftarrow}{\$}$, where the message $\underset{\leftarrow}{\$}$ is recreated to preserve our marking of the tape's end.

Thus, to start a machine, we would use the process

$$\Sigma^\omega \vdash {}^{\triangleleft}\hat{q} \diamond \underset{\leftarrow}{\$} : \Sigma^\omega .$$

Notice that this type can be readily composed. States q and s from different machines could be easily composed with cut:

$$\Sigma^\omega \vdash ({}^{\triangleleft}\hat{q} \diamond \underset{\leftarrow}{\$}) \diamond ({}^{\triangleleft}\hat{s} \diamond \underset{\leftarrow}{\$}) : \Sigma^\omega .$$

This form of composition is sequential, not parallel, because the left-facing head ${}^{\triangleleft}\hat{s}$ must block until the preceding machine terminates and forwards its tape on to ${}^{\triangleleft}\hat{s}$.

Part IV

Relationship between proof construction and proof reduction

10

From processes to rewriting, and back

In the previous chapters, we have presented two different proof-theoretic characterizations of concurrency: *concurrency as proof construction* and *concurrency as proof reduction*. In this chapter, we describe the relationship between these two by identifying fragments of each that can be put into bijective correspondence.

Section 10.1 makes the formula-as-process view of ordered rewriting (chapter 6) formal by embedding session-typed processes (chapter 9) into ordered rewriting. This embedding is execution-preserving, serving as a bisimulation between the proof-reduction and proof-construction approaches to concurrency.

The embedding is quite natural. Process constructors elegantly map to ordered logical connectives, most notably with process composition, $\diamond$, embedding as ordered conjunction, $\bullet$.

(In fact, instead of a single embedding, there are three closely related embeddings – a weakly focused, strongly bisimilar embedding (section 10.1.1); a strongly focused, strongly bisimilar embedding (section 10.1.2); and a strongly focused, weakly bisimilar embedding (section 10.1.3) – that trade off the character of focusing discipline (strong vs. weak) and character of the bisimulation (strong vs. weak) against how directly this correspondence between process constructors and logical connectives is exposed.)

Section 10.1.4 examines the embeddings in the context of the binary counters. Pleasingly, when the example process expressions of section 9.3 are embedded in ordered rewriting, we arrive at exactly the same coinductively defined propositions as used in the choreographies of sections 6.4 and 6.5.

Lastly, and arguably most interestingly, the embedding is, syntactically speaking, an injective mapping, and its left inverse can be leveraged to extract process expressions from (a large subset of) ordered propositions. In section 10.2, we use this idea to reverse-engineer a session-type system for formula-as-process ordered rewriting. Well-typed process expressions embed to propositions that are well-typed under this new system, and vice versa.

This allows us to write global, ordered rewriting specifications of concurrent systems and then extract local, process implementations from them, pro-

vided that those specifications are well-typed. We can have all of the advantages of global specifications, together with all of the advantages of local implementations. At least for the well-typed fragment, proof construction and proof reduction are two sides of the same concurrent coin.

10.1 *Embedding process configurations in formula-as-process ordered rewriting*

Here we give an embedding, $[\![-]\!]$, of process configurations into ordered contexts from the formula-as-process rewriting framework of chapter 6. For the embedding to be adequate, it must preserve the dynamics of processes – the embedding $[\![-]\!]$ must serve as a bisimulation between the reduction semantics for process configurations and the formula-as-process rewriting semantics for ordered contexts.[1]

[1] Because they relate process configuration reductions to ordered rewritings, these bisimulations are not rewriting bisimulations of the kind introduced in chapter 7. Instead, they are ordinary, unlabeled "reduction bisimulations", using the terminology of Sangiorgi and Walker (2003).

Ideally, the embedding $[\![-]\!]$ will be a *strong* bisimulation, so that the correspondence is lockstep. Because the embedding will be a total function, it will be a strong bisimulation if the diagrams

$$\begin{array}{ccc} C & \longrightarrow & C' \\ {\scriptstyle [\![-]\!]}\downarrow & & \downarrow{\scriptstyle [\![-]\!]} \\ [\![C]\!] & \dashrightarrow & [\![C']\!] \end{array} \qquad \text{and} \qquad \begin{array}{ccc} C & \dashrightarrow & C' \\ {\scriptstyle [\![-]\!]}\downarrow & & \vdots{\scriptstyle [\![-]\!]} \\ [\![C]\!] & \longrightarrow & \Omega' \end{array}$$

hold. But strong bisimulations are often elusive, so we will be satisfied to settle for a weak bisimulation if necessary.

In addition, the embedding $[\![-]\!]$ ought to be a monoid homomorphism from process configurations to ordered contexts, so we define

$$[\![C_1 \parallel C_2]\!] = [\![C_1]\!]\,[\![C_2]\!]$$
$$[\![\cdot]\!] = (\cdot)$$

with a clause for $[\![P]\!]$ that remains to be filled in.

By varying how that $[\![P]\!]$ clause is filled in and whether we are working with weak focusing or full focusing (hereafter *strong* focusing), we will arrive at three closely related embeddings: a weakly focused, strongly bisimilar embedding (section 10.1.1); a strongly focused, strongly bisimilar embedding (section 10.1.2); and a strongly focused, weakly bisimilar embedding (section 10.1.3). Of these three embeddings, we prefer the first and last because they most directly expose an appealing correspondence between process constructors and logical connectives.

10.1.1 *A weakly focused, strongly bisimilar embedding*

For the first embedding, we will fill in the $[\![P]\!]$ clause as

$$[\![P]\!] = [P]$$

Process reduction	*Formula-as-process rewriting constraint*
$P \diamond Q \longrightarrow P \parallel Q$	$[P \diamond Q] \dashrightarrow [P]\,[Q]$
$\leftrightarrow \longrightarrow (\cdot)$	$[\leftrightarrow] \dashrightarrow (\cdot)$
$\underset{\rightarrow}{k} \parallel \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow Q_\ell) \longrightarrow Q_k$	$[\underset{\rightarrow}{k}]\,[\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow Q_\ell)] \dashrightarrow [Q_k] \quad (k \in L)$
$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel \underset{\leftarrow}{k} \longrightarrow P_k$	$[\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)]\,[\underset{\leftarrow}{k}] \dashrightarrow [P_k] \quad (k \in L)$

Table 10.1: Constraints on $[-]$ that must be satisfied if $[\![-]\!]$ is to be a strong bisimulation

where $[-]$ is an auxiliary function that embeds process expressions. (For the moment, we will ignore coinductively defined process expressions because they introduce a few complications.)

With this definition for $[\![-]\!]$ in hand, we can then run each process reduction axiom through the first bisimulation diagram to generate constraints on $[-]$ that must be satisfied if $[\![-]\!]$ is to be a strong bisimulation. These axioms and induced constraints are summarized in table 10.1.

For example, the process reduction axiom $P \diamond Q \longrightarrow P \parallel Q$ induces the constraint $[P \diamond Q] \dashrightarrow [P]\,[Q]$. In other words, $[P \diamond Q]$ ought to decompose compositionally in a single step.

We usually presume that the formula-as-process ordered rewriting framework is strongly focused, as described in chapter 6. But suppose that we instead move to a *weakly focused* framework, as sketched in section 5.3.2. Then we can define

$$[P \diamond Q] = [P] \bullet [Q] \quad \text{which satisfies} \quad [P \diamond Q] \dashrightarrow [P]\,[Q]$$

because the weak focusing discipline does not invert positive propositions eagerly. We can also define $[\leftrightarrow] = \mathbf{1}$ which similarly satisfies $[\leftrightarrow] \dashrightarrow (\cdot)$. Notice that there is a clean and direct correspondence between the process constructor $\diamond$ and the logical connective $\bullet$ here; moreover, just as $\leftrightarrow$ is the unit of $\diamond$, so is its embedding the unit of the embedding of $\diamond$.

$$\begin{array}{ccc} P \diamond Q & \longrightarrow & P \parallel Q \\ {\scriptstyle [\![-]\!]}\big| & & \big|{\scriptstyle [\![-]\!]} \\ [\![P \diamond Q]\!] & & [\![P \parallel Q]\!] \\ = & & = \\ [P] \bullet [Q] & \longrightarrow & [P]\,[Q] \end{array}$$

The constraints on $[\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow Q_\ell)]$ can be satisfied by defining

$$[\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow Q_\ell)] = \mathop{\&}_{\ell \in L} \left([\underset{\rightarrow}{\ell}] \setminus \uparrow [Q_\ell]\right).$$

Of course, because the formula-as-process framework restricts left- and right-handed implications to have atomic premises of complementary direction (section 6.1), this suggests that we also define

$$[\underset{\rightarrow}{k}] = \underset{\rightarrow}{k}\,,$$

for otherwise the above left-handed implication will not be a well-formed proposition. Symmetrically, we also define

$$[\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)] = \mathop{\&}_{\ell \in L} \left(\uparrow [P_\ell] \,/\, \underset{\leftarrow}{\ell}\right)$$
$$[\underset{\leftarrow}{k}] = \underset{\leftarrow}{k}\,.$$

Coinductively defined processes require a bit of extra care. As described in section 5.3.2, the weakly focused ordered rewriting framework did not include coinductively defined propositions. Here we assume that coinductively defined negative propositions, like those of the strongly focused formula-as-process ordered rewriting framework (section 6.1.3), also exist for the weakly focused framework.[2]

To embed a process call, $\hat{p}$, in the weakly focused formula-as-process ordered rewriting framework, we will translate that process's definition to the definition of a coinductively defined negative proposition, $\hat{p}^-$. Then the process call is simply embedded as

$$[\hat{p}] = \downarrow \hat{p}^- \,.$$

But how should process *definitions* be translated?

A first, natural attempt would be to map each process definition $\hat{p} \triangleq P$ to a negative proposition definition $\hat{p}^- \triangleq \uparrow[P]$. Unfortunately, this doesn't quite work. It introduces a stutter in the unfolding and rewriting of defined processes like $\hat{p} \triangleq \mathsf{case}_{\ell \in L}(\ell \Rightarrow P_\ell)$. On the process side, we have the single-step

$$\underset{\sim}{k} \parallel \hat{p} = \underset{\sim}{k} \parallel \mathsf{case}_{\ell \in L}(\ell \Rightarrow P_\ell) \longrightarrow P_k \,.$$

But on the ordered rewriting side, it would take two steps to reach the same point:

$$[\![\underset{\sim}{k} \parallel \hat{p}]\!] = \underset{\sim}{k} \downarrow \hat{p}^- = \underset{\sim}{k} \downarrow\uparrow\downarrow \&_{\ell \in L}(\underset{\sim}{\ell} \setminus \uparrow[P_k]) \longrightarrow \underset{\sim}{k} \downarrow \&_{\ell \in L}(\underset{\sim}{\ell} \setminus \uparrow[P_k]) \longrightarrow [P_k] \,,$$

for all $k \in L$. The extra step is caused by the $\uparrow\downarrow$ double shift, with the $\uparrow$ shift arising from the way we are trying to translate definitions, and with the $\downarrow$ shift arising from $[\mathsf{case}_{\ell \in L}(\ell \Rightarrow P_\ell)]$.

Fortunately, this stuttering can be eliminated with a more careful translation of process definitions. Instead of blindly inserting an $\uparrow$ shift in front of $[P]$, let's construct a negative proposition by stripping off the outermost $\downarrow$ shift if $[P]$ has the form $\downarrow A^-$, and otherwise inserting an $\uparrow$ shift:

$$[\![\Xi, (\hat{p} \triangleq P)]\!] = [\![\Xi]\!], \left(\hat{p}^- \triangleq \begin{cases} A^- & \text{if } [P] = \downarrow A^- \\ \uparrow[P] & \text{otherwise} \end{cases}\right)$$
$$[\![\cdot]\!] = (\cdot)$$

Now a defined process like $\hat{p} \triangleq \mathsf{case}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is translated as

$$[\![\hat{p} \triangleq \mathsf{case}_{\ell \in L}(\ell \Rightarrow P_\ell)]\!] = \left(\hat{p}^- \triangleq \&_{\ell \in L}(\underset{\sim}{\ell} \setminus \uparrow[P_\ell])\right) \,,$$

which does not induce the stutter:

$$[\![\underset{\sim}{k} \parallel \hat{p}]\!] = \underset{\sim}{k} \downarrow \hat{p}^- = \underset{\sim}{k} \downarrow \&_{\ell \in L}(\underset{\sim}{\ell} \setminus \uparrow[P_k]) \longrightarrow [P_k] \,,$$

for all $k \in L$.

[2] An argument can be made that the weakly focused framework should include coinductively defined propositions that are only *positive*, not negative, in keeping with its partiality toward positive propositions in stable contexts. That is, however, at odds with the formula-as-process interpretation of coinductively defined propositions as coinductively defined processes, so we choose to have only negative defined propositions. Perhaps this small wrinkle could be ironed out if definitions were treated isorecursively, rather than equirecursively, but we choose not to pursue that here.

In total, the embedding of process configurations and process expressions into a weakly focused formula-as-process ordered rewriting framework is summarized in fig. 10.1.

Notice that, with the exception of a small wrinkle in the translation of process definitions, this embedding is quite clean and direct. Especially appealing is the close correspondence between process constructors and logical connectives noted previously. Moreover, the embedding is a *strong* bisimulation:

Theorem 10.1 (Adequacy of $[\![-]\!]$). *Under the weakly focused formula-as-process ordered rewriting framework, $[\![-]\!]$ constitutes a strong bisimulation. That is:*

- *If $C \longrightarrow C'$, then $[\![C]\!] \longrightarrow [\![C']\!]$.*
- *If $[\![C]\!] \longrightarrow \Omega'^{+}$, then there exists a configuration C' such that $C \longrightarrow C'$ and $[\![C']\!] = \Omega'^{+}$.*

Proof. The first part is by induction on the process configuration reduction, $C \longrightarrow C'$; the second part is by induction on the weakly focused formula-as-process ordered rewriting, $[\![C]\!] \longrightarrow \Omega'^{+}$. □

10.1.2 *A strongly focused, strongly bisimilar embedding*

One might object to the preceding embedding's reliance on weak focusing to achieve strong bisimilarity. Needing to switch from strong focusing to the more exotic weak focusing is admittedly a blemish, but one that is hidden by the appealing correspondence of the process constructors with logical connectives.

If one insists upon strong focusing, there is nevertheless a way forward. Recall from section 5.3.2 that weak focusing can be embedded within strong focusing by the careful addition of shifts. The $(-)^{\square}$ embedding described there could be composed with the $[-]$ embedding of processes described above. For example, the composed embedding of $P_1 \diamond P_2$ would be

$$[P_1 \diamond P_2]^{\square} = \uparrow\big(\bullet([P_1]^{\square}) \bullet \bullet([P_2]^{\square})\big)\,,$$

which satisfies $[P_1 \diamond P_2]^{\square} \longrightarrow [P_1]^{\square}\,[P_2]^{\square}$ in the strong focusing framework. Using such a composition via $[\![P]\!] = [P]^{\square}$ would turn $[\![-]\!]$ into a strongly bisimilar embedding of process configurations within the *strongly* focused formula-as-process ordered rewriting framework. (For a definition of the embedding for process definitions, see fig. 10.2.)

Explicit weak focusing is not needed, as long as we are satisfied with a somewhat more complex embedding that obscures the correspondence between process constructors and logical connectives. However, we do not dwell further on this embedding, instead viewing it as of secondary value because we do prefer the more direct correspondence. Moreover, as we will now show, strong focusing does not exclude a more direct correspondence.

$$[\![C_1 \parallel C_2]\!] = [\![C_1]\!]\,[\![C_2]\!]$$
$$[\![\cdot]\!] = (\cdot)$$
$$[\![P]\!] = [P]$$

and

$$[\![\Xi, (\hat{p} \triangleq P)]\!] = [\![\Xi]\!], \left(\hat{p}^{-} \triangleq \begin{cases} A^{-} & \text{if } [P] = \downarrow A^{-} \\ \uparrow[P] & \text{otherwise} \end{cases}\right)$$
$$[\![\cdot]\!] = (\cdot)$$

where

$$[P_1 \diamond P_2] = [P_1] \bullet [P_2]$$
$$[\leftrightarrow] = \mathbf{1}$$
$$[\underset{\sim}{k}] = \underset{\sim}{k}$$
$$[\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)] = \downarrow \&_{\ell \in L}(\underset{\sim}{\ell} \setminus \uparrow[P_\ell])$$
$$[\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)] = \downarrow \&_{\ell \in L}(\uparrow[P_\ell] \,/\, \underset{\sim}{\ell})$$
$$[\underset{\sim}{k}] = \underset{\sim}{k}$$
$$[\hat{p}] = \downarrow \hat{p}^{-}$$

Figure 10.1: A *strongly* bisimilar embedding of process configurations within a *weakly* focused formula-as-process ordered rewriting framework

$$[\![\cdot]\!] = (\cdot)$$
$$[\![C_1 \parallel C_2]\!] = [\![C_1]\!]\,[\![C_2]\!]$$
$$[\![P]\!] = [P]^{\square}$$

and

$$[\![\Xi, (\hat{p} \triangleq P)]\!] = [\![\Xi]\!], \left(\hat{p}^{-} \triangleq \begin{cases} [P]^{\square} & \text{if } P \neq \underset{\sim}{k}, \underset{\sim}{k} \\ \uparrow[P]^{\square} & \text{otherwise} \end{cases}\right)$$
$$[\![\cdot]\!] = (\cdot)$$

where $[P]$ is defined exactly as in fig. 10.1.

Figure 10.2: A *strongly* bisimilar embedding of process configurations within the *strongly* focused formula-as-process ordered rewriting framework

10.1.3 *A strongly focused, weakly bisimilar embedding*

If we abandon our desire for a strong bisimulation and content ourselves with a weak bisimulation that operates on the strongly focused rewriting framework, then we can, in fact, retain the direct and appealing correspondence with logical connectives. Without changing the definition of $[-]$ at all, let us define $[\![-]\!]$ by

$$\begin{aligned}[\![C_1 \parallel C_2]\!] &= [\![C_1]\!]\,[\![C_2]\!] \\ [\![\cdot]\!] &= (\cdot) \\ [\![P]\!] &= \Omega \text{ where } [[P]] \dashv\Vdash \Omega\,.\end{aligned}$$

Decomposing the process expression $P \diamond Q$ now takes no time at all in its embedded form:

$$P \diamond Q \longrightarrow\!\!\!\!\!\!\longrightarrow P \parallel Q$$

and yet

$$[\![P \diamond Q]\!] = [\![P]\!]\,[\![Q]\!] = [\![P \parallel Q]\!]\,.$$

It is this mismatch that makes this version of $[\![-]\!]$ a weak, not strong, bisimulation.

$$\begin{aligned}[\![C_1 \parallel C_2]\!] &= [\![C_1]\!]\,[\![C_2]\!] \\ [\![\cdot]\!] &= (\cdot) \\ [\![P]\!] &= \Omega \text{ where } [[P]] \dashv\Vdash \Omega\end{aligned}$$

and where $[\![\Xi]\!]$ and $[P]$ are defined exactly as in fig. 10.1.

Figure 10.3: A *weakly* bisimilar embedding of process configurations within the *strongly* focused formula-as-process ordered rewriting framework

Theorem 10.2. *Under the strongly focused formula-as-process ordered rewriting framework, $[\![-]\!]$ constitutes only a* weak *bisimulation. That is:*

- *If $C \longrightarrow C'$, then either $[\![C]\!] \longrightarrow [\![C']\!]$ or $[\![C]\!] = [\![C']\!]$. More specifically, if $C \longrightarrow C'$ arises from the receipt of a message, then $[\![C]\!] \longrightarrow [\![C']\!]$; otherwise, $[\![C]\!] = [\![C']\!]$.*
- *If $[\![C]\!] \longrightarrow \Omega'$, then there exists a configuration C' such that $C \longrightarrow^+ C'$ and $[\![C']\!] = \Omega'$.*

Proof. The first part is by induction on the process configuration reduction, $C \longrightarrow C'$; the second part is by induction on the strongly focused formula-as-process ordered rewriting, $[\![C]\!] \longrightarrow \Omega'$. □

By using the $[-]$ function unchanged and only substituting the clause $[\![P]\!] = [P]$ with $[\![P]\!] = \Omega$ where $[[P]] \dashv\Vdash \Omega$, it is clear that this new version of the $[\![-]\!]$ bisimulation is weak only because positive propositions are now eagerly inverted. That is to say, the weakness of the bisimulation is a purely administrative artifact that is unrelated to the main computational aspects.

10.1.4 *Examples and other comments*

In summary, we have three distinct embeddings of process expressions and configurations within focused formula-as-process ordered rewriting. Focusing, be it weak or strong, is essential to these embeddings. If the unfocused form of ordered rewriting were used, no bisimulation along these lines appears to be possible, not even a weak one. For instance, in a hypothetical

Focusing	*Bisimilarity*	*Key clause*
weakly focused	strongly bisimilar	$[\![P]\!] = [P]$
strongly focused	strongly bisimilar	$[\![P]\!] = [P]^{\square}$
strongly focused	weakly bisimilar	$[\![P]\!] = \Omega$ where $[[P]] \dashv\Vdash \Omega$

Table 10.2: A summary of the process embeddings

unfocused embedding, we would likely have

$$[\![\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)]\!] = \&_{\ell \in L}(\underline{\ell} \setminus [P_\ell]) \longrightarrow \underline{k} \setminus [P_k]$$

if $k \in L$, but there is no configuration C' such that $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \Longrightarrow C'$ and $[\![C']\!] = \underline{k} \setminus [P_k]$.

As examples of the preceding embeddings, let us revisit our two running examples: binary counters and DFAs.

Recall from section 9.3.1 that a binary counter can be implemented by the coinductively defined processes

$$\begin{aligned}\Xi = \big(&e \triangleq \mathsf{caseR}(i \Rightarrow e \diamond b_1 \mid d \Rightarrow \underline{z})\big)\,,\\ &\big(b_0 \triangleq \mathsf{caseR}(i \Rightarrow b_1 \mid d \Rightarrow \underline{d} \diamond b_0')\big)\,,\\ &\big(b_1 \triangleq \mathsf{caseR}(i \Rightarrow \underline{i} \diamond b_0 \mid d \Rightarrow b_0 \diamond \underline{s})\big)\,,\\ &\big(b_0' \triangleq \mathsf{caseL}(z \Rightarrow \underline{z} \mid s \Rightarrow b_1 \diamond \underline{s})\big)\end{aligned}$$

What coinductively defined propositions arise from embedding these process definitions?

Under the strongly focused, weakly bisimilar embedding, the process definition for b_0 becomes the coinductively defined proposition given by

$$[\![b_0 \triangleq \mathsf{caseR}(i \Rightarrow b_1 \mid d \Rightarrow \underline{d} \diamond b_0')]\!] = \big(\hat{b}_0^- \triangleq (\uparrow\downarrow\hat{b}_1^- \,/\, \underline{i}) \,\&\, (\uparrow(\underline{d} \bullet \downarrow\hat{b}_0'^-) \,/\, \underline{d})\big)\,.$$

This defined proposition is *exactly* the same as the one produced in section 6.4.1 as the object-oriented choreography of the counter's initial string rewriting specification. (Here, for completeness, we have included even the minimally necessary shifts that we would usually elide.) Even the $\uparrow\downarrow$ double shift in front of $\hat{b}_1^-$ is correctly produced for free by the embedding.

The same is true of the other process definitions: embedding them yields exactly the same coinductively defined propositions as in the binary counter's object-oriented choreography shown in section 6.4.1.

$$\begin{aligned}[\![\Xi]\!] = \Phi = \big(&\hat{e}^- \triangleq (\uparrow(\downarrow\hat{e}^- \bullet \downarrow\hat{b}_1^-) \,/\, \underline{i}) \,\&\, (\uparrow\underline{z} \,/\, \underline{d})\big)\,,\\ &\big(\hat{b}_0^- \triangleq (\uparrow\downarrow\hat{b}_1^- \,/\, \underline{i}) \,\&\, (\uparrow(\underline{d} \bullet \downarrow\hat{b}_0'^-) \,/\, \underline{d})\big)\,,\\ &\big(\hat{b}_1^- \triangleq (\uparrow(\underline{i} \bullet \downarrow\hat{b}_0^-) \,/\, \underline{i}) \,\&\, (\uparrow(\downarrow\hat{b}_0^- \bullet \underline{s}) \,/\, \underline{d})\big)\,,\\ &\big(\hat{b}_0'^- \triangleq (\underline{z} \setminus \uparrow\underline{z}) \,\&\, (\underline{s} \setminus \uparrow(\downarrow\hat{b}_1^- \bullet \underline{s}))\big)\end{aligned}$$

(Actually, the same definitions arise when using the weakly focused, strongly bisimilar embedding because it treats process definitions in the same way as the strongly focused, weakly bisimilar embedding does.)

Similarly, recall from section 6.4.2 the functional-style process definitions for the binary counter:

$$\begin{aligned}\Xi' = \big(&i \triangleq \mathsf{caseL}(e \Rightarrow \underset{\rightarrow}{e} \diamond \underset{\rightarrow}{b}_1 \mid b_0 \Rightarrow \underset{\rightarrow}{b}_1 \mid b_1 \Rightarrow i \diamond \underset{\rightarrow}{b}_0)\big)\,,\\ &\big(d \triangleq \mathsf{caseL}(e \Rightarrow \underset{\rightarrow}{z} \mid b_0 \Rightarrow d \diamond b'_0 \mid b_1 \Rightarrow \underset{\rightarrow}{b}_0 \diamond \underset{\rightarrow}{s})\big)\,,\\ &\big(b'_0 \triangleq \mathsf{caseL}(z \Rightarrow \underset{\rightarrow}{z} \mid s \Rightarrow \underset{\rightarrow}{b}_1 \diamond \underset{\rightarrow}{s})\big)\,.\end{aligned}$$

By embedding definitions, we arrive at exactly the same coinductively defined propositions as in the functional choreography of the binary counter shown in section 6.4.2:

$$\begin{aligned}[\![\Xi']\!] = \Phi' = \big(&\hat{i}^- \triangleq (\underset{\rightarrow}{e} \setminus \uparrow(\underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1)) \mathbin{\&} (\underset{\rightarrow}{b}_0 \setminus \uparrow\underset{\rightarrow}{b}_1) \mathbin{\&} (\underset{\rightarrow}{b}_1 \setminus \uparrow(\downarrow\hat{i}^- \bullet \underset{\rightarrow}{b}_0))\,,\\ &\big(\hat{d}^- \triangleq (\underset{\rightarrow}{e} \setminus \uparrow\underset{\rightarrow}{z}) \mathbin{\&} (\underset{\rightarrow}{b}_0 \setminus \uparrow(\downarrow\hat{d}^- \bullet \downarrow\hat{b}_0^-)) \mathbin{\&} (\underset{\rightarrow}{b}_1 \setminus \uparrow(\underset{\rightarrow}{b}_0 \bullet \underset{\rightarrow}{s}))\,,\\ &\big(\hat{b}_0'^- \triangleq (\underset{\rightarrow}{z} \setminus \uparrow\underset{\rightarrow}{z}) \mathbin{\&} (\underset{\rightarrow}{s} \setminus \uparrow(\underset{\rightarrow}{b}_1 \bullet \underset{\rightarrow}{s}))\big)\,.\end{aligned}$$

(Once again, the same definitions arise from the weakly focused, strongly bisimilar embedding.)

Although we do not show the details here, the same holds true for DFAs, infinite-word sequential transducers, and even Turing machines: Embedding the process definitions from section 9.3 yields the same coinductively defined propositions as the choreographies from sections 6.5.1 and 6.5.2, in the case of DFAs, or the choreographies that we would have naturally written, in the case of transducers and Turing machines.

10.2 *A session type system for ordered rewriting*

The preceding embeddings describe bisimulations between process expressions and certain ordered propositions; process configurations and certain ordered contexts. Because the formula-as-process ordered rewriting frameworks are untyped, these embeddings discard type information when translating process expressions and configurations. But is that really necessary? Can the bisimilarity witnessed by these embeddings be leveraged to engineer a session type system for formula-as-process ordered rewriting from the session type system for processes?

Consider, for example, the process expression $\underset{\rightarrow}{k}_1 \diamond \underset{\leftarrow}{k}_2$. Although syntactically well-formed, this process expression is not typable: to be typable, the type at the interface between $\underset{\rightarrow}{k}_1$ and $\underset{\leftarrow}{k}_2$ would be faced with the impossible task of simultaneously being both an internal and external choice. Yet even this untypable process expression can be embedded:

$$[\underset{\rightarrow}{k}_1 \diamond \underset{\leftarrow}{k}_2] = \underset{\rightarrow}{k}_1 \bullet \underset{\leftarrow}{k}_2\,.$$

But the question is, can the image of *well-typed* process expressions under this embedding be characterized?

The idea is to engineer a session type system for ordered propositions, based on a judgment $A \vdash A^+ : B$, such that the following theorem will hold.

THEOREM 10.3. *$A \vdash_\Phi A_1^+ : B$ if, and only if, there exist process definitions Ξ and a process expression P such that $A \vdash_\Xi P : B$ and $[\![\Xi]\!] = \Phi$ and $[P] = A_1^+$.*

In the judgment $A \vdash A_1^+ : B$ and this theorem, A and B are (Curry–Howard interpretations of) singleton propositions that function as session types, whereas A_1^+ is a formula-as-process positive ordered proposition that functions as an embedded process expression.

To construct such a type system, we can simply apply the embedding to session typing rules for process expressions. For instance, the typing rule for a process expression $P \diamond Q$ suggests that ordered conjunctions $A_1^+ \bullet A_2^+$ be typed with a cut-like rule, as shown in the adjacent figure. By this procedure, we can also reverse-engineer session-typing rules for the other polarized positive propositions from the session-typing rules for process expressions. We arrive at:

$$\frac{A \vdash A_1^+ : B \quad B \vdash A_2^+ : C}{A \vdash A_1^+ \bullet A_2^+ : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \mathbf{1} : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon\ \ A_\ell \vdash A_\ell^+ : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \downarrow\&_{\ell \in L}(\underset{\rightarrow}{\ell} \setminus \uparrow A_\ell^+) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon\ \ A \vdash A_\ell^+ : B_\ell}{A \vdash \downarrow\&_{\ell \in L}(\uparrow A_\ell^+ \,/\, \underset{\rightarrow}{\ell}) : \&_{\ell \in L}\{\ell : B_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : B_\ell\} \vdash \underset{\leftarrow}{k} : B_k}\ \&\textsc{l}'$$

$$\frac{(A \vdash \hat{p}^- \triangleq A_0^- : B) \in \Phi)}{A \vdash_\Phi \downarrow\hat{p}^- : B}$$

At first, these rules may be a bit startling. It's especially surprising to see the proposition $A_1^+ \bullet A_2^+$ typed without splitting the context and the proposition $\mathbf{1}$ typed with a nonempty context to its left. But once viewed from the formula-as-process perspective, these rules become quite natural: $A_1^+ \bullet A_2^+$ is a composition of process expressions and cut-as-composition is familiar; and $\mathbf{1}$ is a forwarding process, so using the identity rule is not that surprising after all.

Now we can prove the adequacy of these typing rules for positive propositions.

THEOREM 10.3. *$A \vdash_\Phi A_1^+ : B$ if, and only if, there exist process definitions Ξ and a process expression P such that $A \vdash_\Xi P : B$ and $[\![\Xi]\!] = \Phi$ and $[P] = A_1^+$.*

Proof. From left to right, by structural induction on the typing derivation of $A \vdash_\Phi A_1^+ : B$; from right to left, by structural induction on the typing derivation of $A \vdash_\Xi P : B$. □

This would be an extremely strong theorem if not for the specificity of the $\oplus$L and &R rules above. These rules do not assign a type to general negative propositions like $A^- \,\&\, B^-$. Instead, only the restricted propositions $\downarrow\&_{\ell \in L}(\uparrow A_\ell^+ \,/\, \underset{\rightarrow}{\ell})$ and $\downarrow\&_{\ell \in L}(\underset{\rightarrow}{\ell} \setminus \uparrow A_\ell^+)$ are typable. This is because we want

$$\left[\frac{A \vdash P : B \quad B \vdash Q : C}{A \vdash P \diamond Q : C}\right]$$

$$\rightsquigarrow$$

$$\frac{A \vdash [P] : B \quad B \vdash [Q] : C}{A \vdash [P \diamond Q] = [P] \bullet [Q] : C}$$

suggests

$$\frac{A \vdash A_1^+ : B \quad B \vdash A_2^+ : C}{A \vdash A_1^+ \bullet A_2^+ : C}\ \textsc{cut}^B$$

Figure 10.4: The embedding suggests a session-typing rule for ordered conjunction.

these reverse-engineered typing rules to correspond to the image of the process expression typing rules. In general, $A^- \mathbin{\&} B^-$ describes a nondeterministic choice, something that is not present in the syntax of session-typed process expressions.[3] Even so, the theorem is quite strong.

As a final remark, we would like to point out that the 0-ary forms of the above $\oplus$L and $\&$R rules need careful consideration. Because we identify a 0-ary alternative conjunction with $\top$, the 0-ary forms of these rules can lead to a system in which two different typings of $\top$ are possible: $\mathbf{0} \vdash \top : C$ and $A \vdash \top : \top$. For this reason, we disallow 0-ary alternative conjunctions and require that the label set L is nonempty.

[3] This suggests the possibility of introducing a well-typed form of nondeterministic choice. We discuss this idea further in section 11.1.

Coinductively defined propositions can also be given session-typing rules. These take the form of rules verifying that a collection of mutually coinductive definitions is well-typed. The judgment is $\vdash_{\Phi'} \Phi$, where the definitions Φ are judged according to the definitions Φ'. The standard trick of tying the mutually recursive knot by using $\vdash_{\Phi} \Phi$ at the top level is used: Φ is well-typed if $\vdash_{\Phi} \Phi$ holds.

$$\frac{}{\vdash_{\Phi'} (\cdot)} \qquad \frac{(A_0^- \neq \uparrow B_0^+) \quad \vdash_{\Phi'} \Phi \quad A \vdash_{\Phi'} \downarrow A_0^- : B}{\vdash_{\Phi'} \Phi, (A \vdash \hat{p}^- \triangleq A_0^- : B)} \qquad \frac{\vdash_{\Phi'} \Phi \quad A \vdash_{\Phi'} A_0^+ : B}{\vdash_{\Phi'} \Phi, (A \vdash \hat{p}^- \triangleq \uparrow A_0^+ : B)}$$

These rules can be derived by the same kind of reasoning as used for the above proposition typing rules. In particular, the peculiarities of the embedding $[\![\Xi]\!]$ for process definitions explains why the above two rules distinguish cases on whether or not the definition's body begins with an $\uparrow$ shift.

We may prove the following theorem.

Theorem 10.4. *$\vdash_{\Phi'} \Phi$ if, and only if, there exist process definitions Ξ and Ξ' such that $\vdash_{\Xi'} \Xi$ and $[\![\Xi]\!] = \Phi$ and $[\![\Xi']\!] = \Phi'$.*

Proof. From left to right, by structural induction on the typing derivation of $\vdash_{\Phi'} \Phi$; from right to left, by structural induction on $\vdash_{\Xi'} \Xi$. □

Ordered contexts can be given session-typing rules, too. Once again, we can derive these rules by applying the embeddings to the configuration typing rules of section 9.1.3. However, the particulars of ordered contexts differ between the weakly focused and strongly focused frameworks: weakly focused contexts are built from positive propositions, whereas strongly focused contexts are built from negative propositions and positive atoms. So we will actually have two sets of session-typing rules for ordered contexts, using the one that matches the style of focusing in effect.

For the weakly focused framework, the judgment will be $A \Vdash_{\Phi} \Omega^+ : B$, where the definitions Φ are typically elided because they are passed from conclusion to premises unchanged. In the weakly focused framework, the strongly bisimilar embedding yields the following three session-typing rules

for ordered contexts.

$$\frac{A \Vdash \Omega_1^+ : B \quad B \Vdash \Omega_2^+ : C}{A \Vdash \Omega_1^+ \, \Omega_2^+ : C} \text{ C-CUT}^B \qquad \frac{}{A \Vdash \cdot : A} \text{ C-ID}^A$$

$$\frac{A \vdash A_1^+ : B}{A \Vdash A_1^+ : B} \text{ C-PROC}$$

In particular, the final rule can be understood as arising from the weakly focused, strongly bisimilar embedding's clause $[\![P]\!] = [P]$.

THEOREM 10.5. *$A \Vdash_\Phi \Omega^+ : B$ if, and only if, there exists a configuration C such that $A \Vdash_\Xi C : B$ and $[\![\Xi]\!] = \Phi$ and $[\![C]\!] = \Omega^+$.*

Proof. From left to right, by structural induction on the typing derivation of $A \Vdash_\Phi \Omega^+ : B$; from right to left, by structural induction on the typing derivation of $A \Vdash_\Xi C : B$. □

If the strongly focused framework is instead being used, then we will use a different set of session-typing rules for ordered contexts, owing to the fact that ordered contexts are now based on negative propositions and positive atoms, not positive propositions. The strongly focused, weakly bisimilar embedding uses the clause $[\![P]\!] = \Omega$ where $[[P]] \dashv\Vdash \Omega$. Consequently, the following C-PROC rule makes use of the right focus pattern judgment to invert a positive proposition to an ordered context.

$$\frac{A \Vdash \Omega_1 : B \quad B \Vdash \Omega_2 : C}{A \Vdash \Omega_1 \, \Omega_2 : C} \text{ C-CUT}^B \qquad \frac{}{A \Vdash \cdot : A} \text{ C-ID}^A$$

$$\frac{[A^+] \dashv\Vdash \Omega \quad A \vdash A^+ : B}{A \Vdash \Omega : B} \text{ C-PROC}$$

With these rules, we can prove a result of the now familiar form.

THEOREM 10.6. *$A \Vdash_\Phi \Omega : B$ if, and only if, there exists a configuration C such that $A \Vdash_\Xi C : B$ and $[\![\Xi]\!] = \Phi$ and $[\![C]\!] = \Omega$.*

Proof. From left to right, by structural induction on the typing derivation of $A \Vdash_\Phi \Omega : B$; from right to left, by structural induction on $A \Vdash_\Xi C : B$. □

11
Conclusion

In this document, we have explored two proof-theoretic characterizations of concurrency – proof construction and proof reduction – in the context of concurrent systems that have chain topologies.

On the proof-construction side, we have shown a new way of systematically deriving a rewriting framework from the ordered sequent calculus (chapter 5), and identified a message-passing fragment of that framework (chapter 6). We have shown how to take string rewriting specifications of concurrent systems (chapter 4) and choreograph them into message-passing ordered rewriting (chapter 6).

On the proof-reduction side, we have uncovered a semi-axiomatic sequent calculus for singleton logic, a logic that restricts sequents to have exactly one antecedent (chapter 8). We have demonstrated that the semi-axiomatic nature of this calculus gives rise to a clean correspondence between proof normalization and asynchronous message-passing communication (chapter 9).

Lastly, we have shown that the asynchronous processes that arise from the semi-axiomatic sequent calculus for singleton logic can be faithfully embedded within the message-passing ordered rewriting framework (chapter 10). This has provided a relationship between the proof-construction and proof-reduction characterizations of concurrency.

This document now closes by discussing a few avenues for future work.

11.1 *Potential avenues for future work*

11.1.1 *From ordered rewriting to multiset rewriting, singleton processes to linear processes*

One obvious avenue for future work is to extend the ideas in this document to linear logic. We conjecture that the string rewriting specifications of chapter 4 would be replaced with multiset rewriting specifications;[1] the formula-as-process ordered rewriting of chapter 6, with formula-as-process linear rewriting based on the focused linear sequent calculus;[2] and the singleton processes of chapter 9, with Sill processes.[3]

[1] Meseguer 1992.

[2] Miller 1992; Cervesato and Scedrov 2009.

[3] Caires, Pfenning, and Toninho 2016.

For instance, in the formula-as-process ordered rewriting choreography of the binary counter (section 6.4.1), we used the coinductively defined proposition $\hat{b}_0$ given by

$$\hat{b}_0 \triangleq (\uparrow\downarrow\hat{b}_1 \;/\; \underset{\leftarrow}{i}) \;\&\; (\uparrow(\underset{\leftarrow}{d} \bullet \downarrow\hat{b}'_0) \;/\; \underset{\leftarrow}{d})\,,$$

where here all of the shifts have been made explicit. With a move to rewriting based on linear logic, rather than ordered logic, we would likely use the coinductively defined proposition $\hat{b}_0(x, y)$ given by

$$\begin{aligned}\hat{b}_0(x, y) \triangleq\; & \big(\forall y'.\, i(y, y') \multimap \uparrow\downarrow\hat{b}_1(x, y')\big) \\ & \&\; \big(\forall y'.\, d(y, y') \multimap \uparrow\exists x'.\, d(x, x') \otimes \downarrow\hat{b}'_0(x', y')\big)\,,\end{aligned}$$

together with similar defined linear propositions $\hat{e}(x, y)$, $\hat{b}_1(x, y)$, and $\hat{b}'_0(x, y)$. Here the first-order parameters x and y thread together propositions in the context in a way that maintains the binary counter's essential structure. In line with work by Simmons and Pfenning (2011a), $\hat{b}_0(x, y)$ is the *destination-passing* embedding of $\hat{b}_0$. In a formula-as-process interpretation, the destinations x and y can be viewed as channels that connect processes.

This example, however, does not fully exploit the expressive power of first-order linearity because the binary counter still has a chain topology. Because Sill processes admit tree topologies, defined propositions that use destinations in a tree topology should be allowed. But a judgment for checking that destinations do not form cycles would likely be needed, for otherwise a destinations-as-channels interpretation would lead to ill-formed Sill processes.

We would also need to characterize a general procedure for choreographing multiset rewriting specifications, in the vein of what was presented in chapter 6 for string rewriting specifications.

11.1.2 *First-order extension*

Another avenue for future work would be to extend the results contained in this document to first-order, not propositional, ordered rewriting and first-order polymorphic session-typed processes. For example, we might embed sending and receiving processes by

$$\begin{aligned}[a \leftarrow \mathsf{recvR}; P] &= \downarrow\forall a.\, (\uparrow[P] \;/\; \mathsf{t}\underset{\leftarrow}{\mathsf{m}}(a)) \\ [\mathsf{sendL}\; t] &= \mathsf{t}\underset{\leftarrow}{\mathsf{m}}(t)\,,\end{aligned}$$

and similarly for sendR and recvL. It would be nice if the extralogical $\mathsf{t}\underset{\leftarrow}{\mathsf{m}}$ and $\mathsf{t}\underset{\rightarrow}{\mathsf{m}}$ predicates could be done away with, but that appears to be impossible. It seems that, even in an asynchronous calculus, the $\mathsf{t}\underset{\leftarrow}{\mathsf{m}}$ and $\mathsf{t}\underset{\rightarrow}{\mathsf{m}}$ atoms provide the small but necessary amount of synchronization to ensure that the intended term t is used to instantiate the receiving proposition's universal quantifier. However, it does seem plausible that terms could be packaged with the transmission of other labels, such as $\underset{\leftarrow}{k}(t)$, which would hide $\mathsf{t}\underset{\leftarrow}{\mathsf{m}}$ and $\mathsf{t}\underset{\rightarrow}{\mathsf{m}}$.

It would also be interesting to consider how session-typed processes with second-order polymorphism[4] would be embedded in the rewriting framework. A conjecture is that second-order polymorphism would be needed in the rewriting framework so that the embedding could be something like

$$[\alpha \leftarrow \mathsf{recvR}; P] = \downarrow \forall \alpha.\, (\uparrow [P] \;/\; \underset{\leftarrow}{\mathrm{pr}}(\alpha))$$
$$[\mathsf{sendL}\ P] = \underset{\leftarrow}{\mathrm{pr}}([P])\,.$$

[4] Caires, Pérez, et al. 2013.

11.1.3 *Session-typed nondeterministic choice*

In section 10.2, we leveraged the bisimilarity of process expressions and their embedding within formula-as-process ordered rewriting to reverse-engineer a session type system for ordered rewriting. In particular, from the session-typing rules for $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ and $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$, we arrived at rules for typing deterministic choices:

$$\frac{\forall \ell \in L\colon \quad B_\ell \vdash A_\ell^+ : C}{\oplus_{\ell \in L}\{\ell : B_\ell\} \vdash \downarrow \&_{\ell \in L}(\underset{\rightarrow}{\ell} \setminus \uparrow A_\ell^+) : C}\ \oplus\mathrm{L}$$

and

$$\frac{\forall \ell \in L\colon \quad A \vdash A_\ell^+ : B_\ell}{A \vdash \downarrow \&_{\ell \in L}(\uparrow A_\ell^+ \;/\; \underset{\leftarrow}{\ell}) : \&_{\ell \in L}\{\ell : B_\ell\}}\ \&\mathrm{R}$$

Focusing in combination with the left- or right-handed implication ensures that the choices embodied by the alternative conjunction here are deterministic, not nondeterministic, choices.

However, in terms of the ordered propositions alone, it would seem more natural to have a typing rule for $A_1^- \mathbin{\&} A_2^-$ – that is, a rule something like

$$\frac{\vdots}{A \vdash A_1^- \mathbin{\&} A_2^- : C}\,.$$

Finding such a rule for the ordered proposition $A_1^- \mathbin{\&} A_2^-$ might then allow us, by leveraging the ideas behind the bisimilar embedding of chapter 10, to reverse-engineer a process expression for some form of well-behaved, well-typed nondeterministic choice.

Stock[5] has begun to look into incorporating nondeterministic choice into session-typed processes, emphasizing the operational considerations. It would be interesting to consider whether his ideas can be adapted to the formula-as-process ordered rewriting framework.

[5] Stock 2020.

11.1.4 *Induction, coinduction, termination, and productivity*

Derakhshan and Pfenning[6] have developed an infinitary calculus in which inductive and coinductive session types can be used to guarantee the termination and productivity of well-typed processes in a Curry–Howard interpretation of singleton logic. Leveraging types, their validity condition on circular

[6] Derakhshan and Pfenning 2020.

proofs is locally and effectively decidable. Somayyajula and Pfenning[7] have done something similar with sized types. An interesting question is whether their ideas might be applied to (formula-as-process) ordered rewriting.

[7] Somayyajula and Pfenning 2020.

Productivity is about observable progress and even untyped ordered rewriting has a notion of observation, as embodied in ordered bisimilarity (chapter 7). So it seems likely that productive ordered rewriting systems can be characterized.

What is unclear is whether there is an locally and effectively decidable condition on ordered propositions that can guarantee productivity. The local decidability of condition on circular proofs introduced by Derakhshan and Pfenning very much relies on the unrolling of inductive and coinductive types. But ordered rewriting is untyped, at least natively, so whether productivity can be characterized in a locally decidable way is unclear. Also, unlike proofs, rewriting traces are constructed from open-ended derivations. Does that open-endedness in any way affect the existence or shape of the productivity condition?

11.1.5 *Generative invariants and session types*

Simmons[8] describes *generative invariants* as a way to express invariants of ordered logical specifications. These generative invariants generalize context-free grammars, as well as regular worlds from LF. A generative invariant for a binary counter specification in his framework has similarities to the session type for binary counters given in section 9.3.1:

[8] Simmons 2012.

$$\begin{aligned} \mathrm{ctr} &\triangleq e \mathbin{\&} (\mathrm{ctr} \bullet b_0) \mathbin{\&} (\mathrm{ctr} \bullet b_1) \mathbin{\&} (\mathrm{ctr} \bullet i) \\ \mathrm{ctr}' &\triangleq z \mathbin{\&} (\mathrm{ctr} \bullet s) \mathbin{\&} (\mathrm{ctr} \bullet d) \mathbin{\&} (\mathrm{ctr}' \bullet b_0') \\ \kappa &\triangleq \mathbin{\&}\{i : \kappa, d : \oplus\{z : \epsilon, s : \kappa\}\} \end{aligned}$$

It would be interesting to investigate whether these similarities can be extrapolated to a correspondence between generative invariants and session types. If so, generative invariants might serve as a form of session typing for ordered rewriting that is more native than the system presented in section 10.2.

A
Appendix

Theorem A.1. *The following are equivalent.*

- *If* $\Omega\ \mathcal{R}\ (\cdot)$*, then:* $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow (\mathcal{S}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ *for all* $\underset{\rightarrow}{\Delta}$*; and* $\underset{\leftarrow}{\Delta}\ \Omega \Longrightarrow (\underset{\leftarrow}{\Delta}\mathcal{S})\ \underset{\leftarrow}{\Delta}$ *for all* $\underset{\leftarrow}{\Delta}$*.*
- *If* $\Omega\ \mathcal{R}\ (\cdot)$*, then* $\Omega \Longrightarrow (\cdot)\ \mathcal{S}\ (\cdot)$*.*

Proof. Because the premises of the two statements are the same, it suffices to prove that their conclusions are equivalent. We prove each direction separately.

- Assume that $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow (\mathcal{S}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ for all $\underset{\rightarrow}{\Delta}$ and $\underset{\leftarrow}{\Delta}\ \Omega \Longrightarrow (\underset{\leftarrow}{\Delta}\mathcal{S})\ \underset{\leftarrow}{\Delta}$ for all $\underset{\leftarrow}{\Delta}$. Choose an atomic proposition $\underset{\rightarrow}{a}$ that does not appear in Ω; instantiating the emptiness bisimulation condition with $\underset{\rightarrow}{\Delta} = \underset{\rightarrow}{a}$, we have $\underset{\rightarrow}{a}\ \Omega \Longrightarrow (\mathcal{S}\underset{\rightarrow}{a})\ \underset{\rightarrow}{a}$. We can prove by induction on the reduction sequence that $\Omega \Longrightarrow (\cdot)\ \mathcal{S}\ (\cdot)$.
  - Consider the case in which the reduction sequence is trivial, *i.e.*, the case in which $\underset{\rightarrow}{a}\ \Omega\ (\mathcal{S}\underset{\rightarrow}{a})\ \underset{\rightarrow}{a}$. Because $\underset{\rightarrow}{a}$ does not appear in Ω, this holds only if $\Omega = (\cdot)\ \mathcal{S}\ (\cdot)$.
  - Consider the case in which $\underset{\rightarrow}{a}\ \Omega \longrightarrow\Longrightarrow (\mathcal{S}\underset{\rightarrow}{a})\ \underset{\rightarrow}{a}$. Because $\underset{\rightarrow}{a}$ does not appear in Ω, it cannot participate in the initial reduction, so $\Omega \longrightarrow \Omega'$ and $\underset{\rightarrow}{a}\ \Omega' \Longrightarrow (\mathcal{S}\underset{\rightarrow}{a})\ \underset{\rightarrow}{a}$, for some context Ω'. Moreover, because it arises from Ω, the context Ω' does not contain any occurrences of $\underset{\rightarrow}{a}$. From the inductive hypothesis it therefore follows that $\Omega' \Longrightarrow (\cdot)\ \mathcal{S}\ (\cdot)$; prepending the reduction from Ω, we conclude that $\Omega \Longrightarrow (\cdot)\ \mathcal{S}\ (\cdot)$.
- Assume that $\Omega \Longrightarrow (\cdot)\ \mathcal{S}\ (\cdot)$. Because reduction is closed under framing, $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow \underset{\rightarrow}{\Delta}$. Also, $\underset{\rightarrow}{\Delta}\ (\mathcal{S}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$. It follows that $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow (\mathcal{S}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ for all $\underset{\rightarrow}{\Delta}$; and, by symmetric reasoning, that $\Omega\ \underset{\leftarrow}{\Delta} \Longrightarrow (\underset{\leftarrow}{\Delta}\mathcal{S})\ \underset{\leftarrow}{\Delta}$ for all $\underset{\leftarrow}{\Delta}$. □

Bibliography

Abramsky, Samson (1993)
Computational Interpretations of Linear Logic. In: *Theoretical Computer Science* 111.1–2, pp. 3–57 (cit. on p. 9).

Abrusci, Vito Michele (1990)
Non-Commutative Intuitionistic Linear Logic. In: *Mathematical Logic Quarterly* 36.4, pp. 297–318 (cit. on pp. 11, 12, 27, 38, 55).

Amadio, Roberto M. and Luca Cardelli (1993)
Subtyping Recursive Types. In: *ACM Transactions on Programming Languages and Systems* 15.4, pp. 575–631 (cit. on p. 77).

Amadio, Roberto M., Ilaria Castellani, and Davide Sangiorgi (1998)
On Bisimulations for the Asynchronous pi-Calculus. In: *Theoretical Computer Science* 195.2, pp. 291–324 (cit. on pp. 104, 106).

Andreoli, Jean-Marc (1992)
Logic Programming with Focusing Proofs in Linear Logic. In: *Journal of Logic and Computation* 2.3 (June 1, 1992), pp. 297–347 (cit. on pp. 9, 12, 56, 62).

Avron, Arnon (1988)
The Semantics and Proof Theory of Linear Logic. In: *Theoretical Computer Science* 57.2–3. Ed. by Maurice Nivat, pp. 161–184 (cit. on p. 132).

Baelde, David and Dale Miller (2007)
Least and Greatest Fixed Points in Linear Logic. In: *Proceedings of the 14th International Conference on Logic for Programming, Artificial Intelligence, and Reasoning* (Yerevan, Armenia, Oct. 15–19, 2007). Ed. by Nachum Dershowitz and Andrei Voronkov. Vol. 4790. Springer, pp. 92–106 (cit. on p. 77).

Balzer, Stephanie and Frank Pfenning (2015)
Objects as Session-Typed Processes. In: *Proceedings of the 5th International Workshop on Programming Based on Actors, Agents, and Decentralized Control* (Pittsburgh, PA, USA, Oct. 26, 2015). Ed. by Elisa Gonzalez Boix, Philipp Haller, Alessandro Ricci, and Carlos Varela. New York: ACM, pp. 13–24 (cit. on p. 90).

Béal, Marie-Pierre and Olivier Carton (2002)
Determinization of Transducers over Finite and Infinite Words. In: *Theoretical Computer Science* 289.1, pp. 225–251 (cit. on p. 24).

Bellin, Gianluigi and Philip J. Scott (1994)
On the pi-Calculus and Linear Logic. In: *Theoretical Computer Science* 135.1, pp. 11–65 (cit. on p. 9).

Benton, P. Nick (1995)
A Mixed Linear and Non-Linear Logic. Proofs, Terms and Models. In: *Selected Papers from the 8th International Workshop on Computer Science Logic* (Kazimierz, Poland, Sept. 25–30, 1994). Ed. by Leszek Pacholski and Tiuryn Jerzy. Vol. 933. Lecture Notes in Computer Science. Springer, pp. 121–135 (cit. on p. 39).

Boreale, Michele, Rocco De Nicola, and Rosario Pugliese (2002)
Trace and Testing Equivalence on Asynchronous Processes. In: *Information and Computation* 172.2, pp. 139–164 (cit. on pp. 104, 106).

Caires, Luís, Jorge A. Pérez, Frank Pfenning, and Bernardo Toninho (2013)
Behavioral Polymorphism and Parametricity in Session-Based Communication. In: *Proceedings of the 22nd European Symposium on Programming* (Rome, Italy, Mar. 16–24, 2013). Ed. by Matthias Felleisen and Philippa Gardner. Vol. 7792. Lecture Notes in Computer Science. Springer, pp. 330–349 (cit. on p. 173).

Caires, Luís and Frank Pfenning (2010)
Session Types as Intuitionistic Linear Propositions. In: *CONCUR 2010 - Concurrency Theory, 21st International Conference* (Paris, France, Aug. 31–Sept. 3, 2010). Ed. by Paul Gastin and François Laroussinie. Vol. 6269. Lecture Notes in Computer Science. Springer (cit. on pp. 9, 15, 125, 145).

Caires, Luís, Frank Pfenning, and Bernardo Toninho (2012)
Towards Concurrent Type Theory. In: *Proceedings of the 8th ACM SIGPLAN Workshop on Types in Languages Design and Implementation* (Philadelphia, Pennsylvania, USA, Jan. 28, 2012). Ed. by Benjamin C. Pierce. TLDI '12. New York: ACM, pp. 1–12 (cit. on pp. 9, 15, 125, 141, 145).

Caires, Luís, Frank Pfenning, and Bernardo Toninho (2016)
Linear Logic Propositions as Session Types. In: *Mathematical Structures in Computer Science* 26.3: *Special Issue. Behavioral Types, Part 2*. Ed. by Simon J. Gay and António Ravara, pp. 367–423 (cit. on pp. 9–11, 15, 125, 141, 145, 171).

Cervesato, Iliano, Nancy Durgin, Patrick D. Lincoln, John C. Mitchell, and Andre Scedrov (1999)
A Meta-Notation for Protocol Analysis. In: *Proceedings of the 12th IEEE*

Computer Security Foundations Workshop (Mordano, Italy, June 28–30, 1999). IEEE Computer Society Press, pp. 55–69 (cit. on p. 43).

Cervesato, Iliano, Joshua S. Hodas, and Frank Pfenning (2000)
Efficient Resource Management for Linear Logic Proof Search. In: *Theoretical Computer Science* 232.1–2 (Feb. 2000): *Special Issue. Proof-search in Type-theoretic Languages*. Ed. by Didier Galmiche and David J. Pym, pp. 133–163 (cit. on p. 127).

Cervesato, Iliano and Andre Scedrov (2009)
Relating State-Based and Process-Based Concurrency through Linear Logic. In: *Information and Computation* 207.10 (Oct. 2009): *Special Issue. 13th Workshop on Logic, Language, Information and Computation (WoLLIC 2006)*. Ed. by Grigori Mints, Valéria de Paiva, and Ruy de Queiroz, pp. 1044–1077 (cit. on pp. 10, 13, 43, 62, 69, 171).

Chang, Bor-Yuh Evan, Kaustuv Chaudhuri, and Frank Pfenning (2003)
A Judgmental Analysis of Linear Logic. Tech. rep. CMU-CS-03-131R. Carnegie Mellon University, Computer Science Department, Dec. 2003 (cit. on pp. 30, 39).

Cormen, Thomas H., Charles E. Leiserson, Ronald L. Rivest, and Clifford Stein (2009)
Introduction to Algorithms. 3rd ed. The MIT Press (cit. on p. 51).

Curry, Haskell B. (1934)
Functionality in Combinatory Logic. In: *Proceedings of the National Academy of Sciences of the United States of America* 20.11 (Nov. 1934), pp. 584–590 (cit. on p. 9).

Deng, Yuxin, Robert J. Simmons, and Iliano Cervesato (2016)
Relating Reasoning Methodologies in Linear Logic and Process Algebra. In: *Mathematical Structures in Computer Science* 26.5, pp. 868–906 (cit. on pp. 14, 105, 108).

Derakhshan, Farzaneh and Frank Pfenning (2020)
Circular Proofs as Session-Typed Processes. A Local Validity Condition. In: *Logical Methods in Computer Science* (Aug. 12, 2020). arXiv: 1908.01909. Submitted (cit. on pp. 10, 16, 17, 38, 150, 152, 173, 174).

DeYoung, Henry, Luís Caires, Frank Pfenning, and Bernardo Toninho (2012)
Cut Reduction in Linear Logic as Asynchronous Session-Typed Communication. In: *Proceedings of the 21st Annual Conference of the EACSL on Computer Science Logic* (Fontainebleau, France, Sept. 3–6, 2012). Ed. by Patrick Cégielski and Arnaud Durand. Vol. 16. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 228–242 (cit. on pp. 16, 142).

DeYoung, Henry and Frank Pfenning (2016)
Substructural Proofs as Automata. In: *Proceedings of the 14th Asian Symposium on Programming Languages and Systems* (Hanoi, Vietnam, Nov. 21–23, 2016). Ed. by Atsushi Igarashi. Vol. 10017. Lecture Notes in Computer Science. Invited talk. Springer, Nov. 2016, pp. 3–22 (cit. on p. 153).

DeYoung, Henry, Frank Pfenning, and Klaas Pruiksma (2020)
Semi-Axiomatic Sequent Calculus. In: *5th International Conference on Formal Structures for Computation and Deduction* (Paris, France, June 29–July 6, 2020). Ed. by Zena M. Ariola. Vol. 167. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, 29:1–29:22 (cit. on pp. 15, 124, 131).

Dezani-Ciancaglini, Mariangiola, Luca Padovani, and Jovanka Pantovic (2014)
Session Type Isomorphisms. In: *Proceedings of the 7th Workshop on Programming Language Approaches to Concurrency and Communication-cEntric Software* (Grenoble, France, Apr. 12, 2014). Ed. by Alastair F. Donaldson and Vasco T. Vasconcelos. Vol. 155. EPTCS, pp. 61–71 (cit. on pp. 16, 141).

Dummett, Michael (1976)
The William James Lectures. Later published as: Michael Dummett (1991) *The Logical Basis of Metaphysics*. Cambridge, Massachusetts: Harvard University Press (cit. on pp. 27, 33, 128).

Durgin, Nancy, Patrick Lincoln, John C. Mitchell, and Andre Scedrov (2004)
Multiset Rewriting and the Complexity of Bounded Security Protocols. In: *Journal of Computer Security* 12.2 (Feb. 1, 2004), pp. 247–311 (cit. on p. 43).

Eriksson, Lars-Henrik (1991)
A Finitary Version of the Calculus of Partial Inductive Definitions. In: *Proceedings of the Second International Workshop on Extensions of Logic Programming* (Stockholm, Sweden, Jan. 27–29, 1991). Ed. by Lars-Henrik Eriksson, Lars Hallnäs, and Peter Schroeder-Heister. Vol. 596. Lecture Notes in Computer Science. Springer, pp. 89–134 (cit. on pp. 38, 76).

Felleisen, Matthias and Philippa Gardner, eds. (2013)
Proceedings of the 22nd European Symposium on Programming (Rome, Italy, Mar. 16–24, 2013). Vol. 7792. Lecture Notes in Computer Science. Springer.

Fortier, Jérôme and Luigi Santocanale (2013)
Cuts for Circular Proofs: Semantics and Cut-Elimination. In: *Proceedings of the 22nd Annual Conference of the EACSL on Computer Science Logic*. Ed. by Simona Ronchi Della Rocca. Vol. 23. Leibniz International Proceedings in Informatics (LIPIcs). Dagstuhl, Germany: Schloss Dagstuhl–Leibniz-Zentrum für Informatik, pp. 248–262 (cit. on pp. 11, 14, 38, 76, 138).

Gay, Simon J. and Malcolm Hole (2005)
Subtyping for Session Types in the Pi Calculus. In: *Acta Informatica* 42.2–3, pp. 191–225 (cit. on pp. 76, 149).

Gentzen, Gerhard (1935)
Untersuchungen über das logische Schließen. In: *Mathematische Zeitschrift* 39, pp. 176–210, 405–431. English translation in: M. E. Szabo, ed. (1969) *The Collected Papers of Gerhard Gentzen.* North-Holland, pp. 68–131 (cit. on pp. 27, 33, 123, 128).

Ginsburg, Seymour and Gene F. Rose (1966)
A Characterization of Machine Mappings. In: *Canadian Journal of Mathematics* 18, pp. 381–388 (cit. on p. 24).

Girard, Jean-Yves (1987)
Linear Logic. In: *Theoretical Computer Science* 50.1, pp. 1–101 (cit. on pp. 9, 27, 28, 38).

Griffith, Dennis E. (2016)
Polarized Substructural Session Types. PhD thesis. University of Illinois at Urbana-Champaign (cit. on p. 10).

Hallnäs, Lars (1991)
Partial Inductive Definitions. In: *Theoretical Computer Science* 87.1, pp. 115–142 (cit. on pp. 38, 76).

Harper, Robert (2011)
The Holy Trinity. Published in the *Existential Type* weblog. Mar. 27, 2011 (cit. on pp. 123, 124).

Honda, Kohei (1993)
Types for Dyadic Interaction. In: *Proceedings of the 4th International Conference on Concurrency Theory* (Hildesheim, Germany, Aug. 23–26, 1993). Ed. by Eike Best. Vol. 715. Lecture Notes in Computer Science. Springer, pp. 509–523 (cit. on p. 9).

Honda, Kohei, Nobuko Yoshida, and Marco Carbone (2008)
Multiparty Asynchronous Session Types. In: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, Jan. 7–12, 2008). Ed. by George C. Necula and Philip Wadler. New York: ACM, pp. 273–284 (cit. on p. 17).

Hopcroft, John E., Rajeev Motwani, and Jeffrey D. Ullman (2006)
Introduction to Automata Theory, Languages, and Computation. 3rd ed. Pearson (cit. on pp. 21, 25).

Howard, William A. (1969)
The Formulae-as-Types Notion of Construction. Unpublished note. An annotated version appeared in: J. Roger Hindley and Jonathan P. Seldin, eds.

(1980)
To H. B. Curry. Essays on Combinatory Logic, Lambda Calculus, and Formalism. Academic Press (cit. on p. 9).

Kanazawa, Makoto (1992)
The Lambek Calculus Enriched with Additional Connectives. In: *Journal of Logic, Language, and Information* 1.2 (June 1, 1992), pp. 141–171 (cit. on pp. 11, 12, 27, 55).

Kanovich, Max, Stepan Kuznetsov, Vivek Nigam, and Andre Scedrov (2019)
Subexponentials in Non-Commutative Linear Logic. In: *Mathematical Structures in Computer Science* 29.8, pp. 1217–1249 (cit. on p. 39).

Kleene, Stephen C. (1952)
Introduction to Metamathematics. North-Holland (cit. on p. 123).

Lambek, Joachim (1958)
The Mathematics of Sentence Structure. In: *The American Mathematical Monthly* 65.3, pp. 154–170 (cit. on pp. 11, 12, 27, 55, 57, 69).

Lambek, Joachim (1961)
On the Calculus of Syntactic Types. In: *Structure of Language and Its Mathematical Aspects*. Ed. by R. Jakobson, pp. 166–178 (cit. on pp. 11, 12, 27, 55).

Laurent, Olivier (2002)
Étude de la polarisation en logique. French. PhD thesis. Université Aix-Marseille II, Mar. 2002 (cit. on p. 67).

Maehara, Shôji (1954)
Eine Darstellung der intuitionistischen Logik in der klassischen. In: *Nagoya Mathematical Journal* 7, pp. 45–64 (cit. on p. 123).

Martin-Löf, Per (1982)
Constructive Mathematics and Computer Programming. In: *Logic, Methodology and Philosophy of Science VI*. Ed. by L. Jonathan Cohen, Jerzy Łoś, Helmut Pfeiffer, and Klaus-Peter Podewski. Vol. 104. Studies in Logic and the Foundations of Mathematics. Elsevier, pp. 153–175 (cit. on p. 9).

Martin-Löf, Per (1983)
Siena Lectures. Transcript later published as: Per Martin-Löf (1996)
On the Meanings of the Logical Constants and the Justifications of the Logical Laws. In: *Nordic Journal of Philosophical Logic* 1.1, pp. 11–60. Apr. 6–9, 1983 (cit. on pp. 27, 33, 128).

McDowell, Raymond and Dale Miller (2000)
Cut-elimination for a Logic with Definitions and Induction. In: *Theoretical Computer Science* 232.1–2, pp. 91–119 (cit. on pp. 38, 76).

Mendler, Nax P. (1987)
Recursive Types and Type Constraints in Second-Order Lambda Calculus. In: *Proceedings of the Second Annual IEEE Symposium on Logic in Computer Science* (Ithaca, NY, USA, June 22–25, 1987). IEEE Computer Society Press, pp. 30–36 (cit. on p. 10).

Meseguer, José (1992)
Conditional Rewriting Logic as a Unified Model of Concurrency. In: *Theoretical Computer Science* 96.1, pp. 73–155 (cit. on pp. 12, 43, 171).

Miller, Dale (1992)
The pi-Calculus as a Theory in Linear Logic. Preliminary Results. In: *Proceedings of the Third International Workshop on Extensions of Logic Programming* (Bologna, Italy, Feb. 26–28, 1992). Ed. by Evelina Lamma and Paola Mello. Vol. 660. Lecture Notes in Computer Science. Springer, pp. 242–264 (cit. on pp. 10, 13, 69, 171).

Miller, Dale, Gopalan Nadathur, Frank Pfenning, and Andre Scedrov (1991)
Uniform Proofs as a Foundation for Logic Programming. In: *Annals of Pure and Applied Logic* 51.1–2 (Mar. 14, 1991), pp. 125–157 (cit. on p. 9).

Milner, Robin (1999)
Communicating and Mobile Systems. The Pi-Calculus. 1st ed. Cambridge University Press, May 20, 1999 (cit. on pp. 38, 76).

Milner, Robin, Joachim Parrow, and David Walker (1992a)
A Calculus of Mobile Processes, I. In: *Information and Computation* 100.1, pp. 1–40 (cit. on p. 38).

Milner, Robin, Joachim Parrow, and David Walker (1992b)
A Calculus of Mobile Processes, II. In: *Information and Computation* 100.1, pp. 41–77 (cit. on p. 38).

Necula, George C. and Philip Wadler, eds. (2008)
Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages (San Francisco, California, Jan. 7–12, 2008). New York: ACM.

Nigam, Vivek and Dale Miller (2009)
Algorithmic Specifications in Linear Logic with Subexponentials. In: *Proceedings of the 11th International ACM SIGPLAN Conference on Principles and Practice of Declarative Programming* (Coimbra, Portugal, Sept. 7–9, 2009). Ed. by António Porto and Francisco Javier López-Fraguas. New York: ACM, pp. 129–140 (cit. on p. 39).

Pfenning, Frank (1995)
Structural Cut Elimination. In: *Proceedings of the 10th Annual IEEE Symposium on Logic in Computer Science* (San Diego, California, June 26–29,

1995). Ed. by Dexter Kozen. IEEE Computer Society Press, pp. 156–166 (cit. on p. 34).

Pfenning, Frank (2008)
Church and Curry. Combining Intrinsic and Extrinsic Typing. In: *Reasoning in Simple Type Theory. Festschrift in Honor of Peter B. Andrews on His 70th Birthday*. Ed. by Christoph Benzmüller, Chad Brown, Jörg. Siekmann, and Richard Statman. Vol. 17. Studies in Logic. College Publications, pp. 303–338 (cit. on p. 128).

Pfenning, Frank (2016)
Lecture Notes from 15-816: Substructural Logics. Carnegie Mellon University, Computer Science Department, Aug. 30–Dec. 8, 2016 (cit. on p. 27).

Pfenning, Frank and Robert J. Simmons (2009)
Substructural Operational Semantics as Ordered Logic Programming. In: *Proceedings of the 24th Annual IEEE Symposium on Logic in Computer Science* (Los Angeles, CA, Aug. 11–14, 2009). IEEE Computer Society Press, pp. 101–110 (cit. on p. 49).

Pierce, Benjamin C. (2002)
Types and Programming Languages. MIT Press (cit. on p. 77).

Polakow, Jeff (2001)
Ordered Linear Logic and Applications. PhD thesis. Carnegie Mellon University (cit. on pp. 12, 13, 76).

Polakow, Jeff and Frank Pfenning (1999a)
Natural Deduction for Intuitionistic Non-Commutative Linear Logic. In: *Proceedings of the 4th International Conference on Typed Lambda Calculi and Applications* (L'Aquila, Italy, Apr. 7–9, 1999). Ed. by Jean-Yves Girard. Vol. 1581. Lecture Notes in Computer Science. Springer, pp. 295–309 (cit. on p. 38).

Polakow, Jeff and Frank Pfenning (1999b)
Relating Natural Deduction and Sequent Calculus for Intuitionistic Non-Commutative Linear Logic. In: *15th Conference on Mathematical Foundations of Progamming Semantics* (New Orleans, LA, USA, Apr. 28–May 1, 1999). Ed. by Stephen D. Brookes, Achim Jung, Michael W. Mislove, and Andre Scedrov. Vol. 20. Electronic Notes in Theoretical Computer Science. Elsevier, pp. 449–466 (cit. on pp. 11, 27, 34, 36, 38, 39, 55).

Pous, Damien and Davide Sangiorgi (2011)
Enhancements of the Bisimulation Proof Method. In: *Advanced Topics in Bisimulation and Coinduction*. Ed. by Davide Sangiorgi and Jan Rutten. Cambridge Tracts in Theoretical Computer Science. Cambridge University Press, pp. 233–289 (cit. on p. 108).

Pruiksma, Klaas, William Chargin, Frank Pfenning, and Jason Reed (2018)
Adjoint Logic. Unpublished manuscript. Apr. 2018 (cit. on p. 39).

Sambin, Giovanni, Giulia Battilotti, and Claudia Faggian (2000)
Basic Logic: Reflection, Symmetry, Visibility. In: *Journal of Symbolic Logic* 65.3, pp. 979–1013 (cit. on pp. 138, 139).

Sangiorgi, Davide and David Walker (2003)
The Pi-Calculus: A Theory of Mobile Processes. Cambridge University Press, Oct. 16, 2003 (cit. on pp. 14, 66, 108, 160).

Santocanale, Luigi (2002)
A Calculus of Circular Proofs and Its Categorical Semantics. In: *Proceedings of the 5th International Conference on Foundations of Software Science and Computation Structures* (Grenoble, France, Apr. 8–12, 2002). Ed. by Mogens Nielsen and Uffe Engberg. Vol. 2303. Lecture Notes in Computer Science. Springer, pp. 357–371 (cit. on pp. 11, 14, 138).

Schack-Nielsen, Anders and Carsten Schürmann (2008)
Celf. A Logical Framework for Deductive and Concurrent Systems (System Description). In: *Proceedings of the 4th International Joint Conference on Automated Reasoning* (Sydney, Australia, Aug. 12–15, 2008). Ed. by Alessandro Armando, Peter Baumgartner, and Gilles Dowek. Vol. 5195. Lecture Notes in Computer Science. Springer, pp. 320–326 (cit. on p. 127).

Schroeder-Heister, Peter (1993)
Rules of Definitional Reflection. In: *Proceedings of the Eighth Annual IEEE Symposium on Logic in Computer Science* (Montreal, Canada, June 19–23, 1993). IEEE Computer Society Press, pp. 222–232 (cit. on pp. 38, 76).

Schützenberger, Marcel-Paul (1977)
Sur une variante des fonctions séquentielles. French. In: *Theoretical Computer Science* 4.1, pp. 47–57 (cit. on p. 24).

Simmons, Robert J. (2012)
Substructural Logical Specifications. PhD thesis. Carnegie Mellon University, Computer Science Department, Nov. 2012 (cit. on pp. 12, 17, 56, 62, 64, 76, 174).

Simmons, Robert J. and Frank Pfenning (2011a)
Logical Approximation for Program Analysis. In: *Higher Order Symbolic Computation* 24.1–2, pp. 41–80 (cit. on p. 172).

Simmons, Robert J. and Frank Pfenning (2011b)
Weak Focusing for Ordered Linear Logic. Tech. rep. CMU-CS-10-147. Carnegie Mellon University, Computer Science Department, Apr. 2011 (cit. on p. 67).

Somayyajula, Siva and Frank Pfenning (2020). Personal communication (cit. on pp. 10, 16, 17, 150, 152, 174).

Steinberger, Florian (2011)
Why Conclusions Should Remain Single. In: *Journal of Philosophical Logic* 40.3 (June 2011), pp. 333–355 (cit. on p. 123).

Stock, Benedikt (2020)
General Pattern Matching for Session-Typed Concurrent Programs. Bachelor thesis. Jacobs University Bremen, May 15, 2020 (cit. on p. 173).

Tiu, Alwen and Alberto Momigliano (2012)
Cut Elimination for a Logic with Induction and Co-induction. In: *Journal of Applied Logic* 10.4, pp. 330–367 (cit. on pp. 38, 76).

Toninho, Bernardo (2015)
A Logical Foundation for Session-Based Concurrent Computation. PhD thesis. Universidade Nova de Lisboa, May 2015 (cit. on p. 9).

Toninho, Bernardo, Luís Caires, and Frank Pfenning (2013)
Higher-Order Processes, Functions, and Sessions. A Monadic Integration. In: *Proceedings of the 22nd European Symposium on Programming* (Rome, Italy, Mar. 16–24, 2013). Ed. by Matthias Felleisen and Philippa Gardner. Vol. 7792. Lecture Notes in Computer Science. Springer, pp. 350–369 (cit. on pp. 10, 125, 145).

Watkins, Kevin, Iliano Cervesato, Frank Pfenning, and David Walker (2002)
A Concurrent Logical Framework I. Judgments and Properties. Tech. rep. CMU-CS-02-101. Department of Computer Science, Carnegie Mellon University. Revised May 2003 (cit. on pp. 12, 13, 46, 62, 76).

Zeilberger, Noam (2008)
Focusing and Higher-Order Abstract Syntax. In: *Proceedings of the 35th ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (San Francisco, California, Jan. 7–12, 2008). Ed. by George C. Necula and Philip Wadler. New York: ACM, pp. 359–369 (cit. on pp. 12, 56, 63).