HENRY DEYOUNG

SESSION-TYPED ORDERED LOGICAL SPECIFICATIONS

Contents

List of Figures

List of Tables

1
Introduction

- Computation as deduction: clear, expressive, and provably correct programs
  - Examples of sucess stories
  - Can it be applied to concurrency?
- Proof constuction and proof reduction views of con currency
  - Proof construction: good for specifications
  - Proof reduction: good for implementions
- Thesis statement: Session types bridge these two views
- Ordered logic as a proving ground
  - Ordered rewriting for proof construction
  - Singleton logic (purely additive fragment of ordered logic) for proof reduction
- Ordered rewriting (chapter 4) for specifications
  - DFAs and NFAs
  - Binary counters
- Refinement of ordered rewriting for choreographies (chapter 5)
  - Recursive definitions as processes; atoms as messages
  - Untyped (mostly, except for directions)
  - Rewriting bisimilarity for observational equivalence
    - Examples
- Singleton logic and its semi-axiomatic calculus (chapter 6)
-

Concurrent systems are notoriously difficult to get right.

Beginning with Curry's observation that Hilbert [...] corresponds to a form of computation based on combinatory reduction,[1] and continuing with Howard's discovery of an isomorphism between [Gentzen's] intuitionistic natural deduction and Church's simply-typed λ-calculus, computation-as-deduction has been the gold standard for clear, expressive, and provably correct programs.

[1] ??.

Computation-as-deduction can be divided into two classes: proof-search-as-computation and proof-reduction-as-computation. The former provides a logically grounded basis for the backward- and forward-chaining logic programming paradigms, whereas the latter is the foundation for the functional programming paradigm.

Logically grounded concurrent computation

More recently, a proof-reduction description of concurrency has been discovered by ?? with ??. In this isomorphism, linear propositions correspond to session types; sequent proofs, to session-typed processes; and cut reduction, to synchronous message-passing communication.

This thesis seeks to bring these two apparently divergent views of concurrency together. Is there a class of specifications for which well-typed implementations can automatically be extracted?

Thesis statement: Session types form the bridge.

1.1 *Proposal 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, concomitant with nondeterminism, concurrent systems are notoriously tricky to get right: subtle races and deadlocks can occur even in the most rigorously tested of systems.

At the same time, decades of research into connections between proof theory and programming languages have firmly established the principle of *computation as deduction* as the gold standard [framework] for clear, expressive, and provably correct programs. Most generally, intuitionistic logic is the bedrock for both the typed functional[2] and logic programming[3] [4] paradigms. In more specific [...] Examples abound: lax logic for effectful computation,[5] temporal logic for functional reactive programming,[6] and linear logic for graph-based algorithms,[7] to name just a few.

[2] **Martin-Lof:LMPS80**.
[3] **Miller+:PAL91Andreoli:JLC92**.
[4] check refs?
[5] **Benton:JFP98**.
[6] **Jeffrey:PLPV12**.
[7] **Cruz+:ICLP14**.

Can a computation-as-deduction approach make it similarly easier to clearly and concisely specify, as well as correctly implement, concurrent programs?

1.1.1

The principle of computation as deduction comes in two flavors: *proof construction as computation* and *proof reduction as computation*. Under a proof-construction-as-computation view, the search for a proof, according to a fixed strategy, forms the basis of computation; it is the foundation for logic programming.[8] The proof-reduction-as-computation view, on the other hand, revolves around a correspondence, known as the Curry–Howard Isomorphism[9] between propositions and types, proofs and programs, and proof simplification, or reduction, and program evaluation; it is the foundation for typed functional programming.[10]

[8] **Miller+:PAL91Andreoli:JLC92**.

[9] Howard22:??.

[10] **Martin-Lof:LMPS80**.

Both the proof-construction and proof-reduction approaches have been successfully applied to concurrent programming, originally stemming from Girard's suggestion of connections between linear logic and concurrency.[11] In the proof-construction vein, the Concurrent Logical Framework (CLF)[12] treats the permutability of inference rules as the source of concurrency. CLF has been used to specify a variety of concurrent systems, ranging from the π-calculus to security protocols and even emergent story narratives.[13] [14] Although these same concurrent systems can be simulated according to their CLF specifications by the Lollimon[15] and Celf[16] logic programming engines, the programs ultimately remain specifications, not actual *decentralized* implementations.

[11] Girard 1987.

[12] **Watkins+:CMU02**.

[13] Cervesato and Scedrov 2009**Martens+:LPNMR13**.

[14] check refs

[15] **Lopez+:PPDP05**.

[16] **Schack-Nielsen:ITU11**.

Taking the other, proof-reduction tack, **Abramsky:TCS93**, **Bellin+Scott:TCS94**, and later **Caires+Pfenning:CONCUR10** with Toninho[17] have given correspondences between sequent calculus proofs or proof nets in linear logic and processes; between cut elimination and concurrent process execution. Moreover, in **Caires+:MSCS13**'s work, the correspondence is a true Curry–Howard isomorphism in that intuitionistic linear propositions are also types – *session types*[18] that describe the interaction protocol to which a process adheres. Unlike proof construction, the proof-reduction approach yields actual decentralized implementations with independent threads of control.[19]

[17] Caires et al. 2012**Caires+:MSCS13**.

[18] **Honda:CONCUR93**.

[19] **Toninho+:ESOP13Griffith+Pfenning:14**.

In spite of their common basis in linear logic, these proof-construction and proof-reduction approaches to concurrent computation appear at first glance to be strikingly disparate. They have different dynamics; they offer different guarantees (session fidelity, behavioral type preservation, and deadlock freedom for the proof-reduction approach, but only non-behavioral type preservaton for the proof-construction approach); and, perhaps most importantly, they serve very different roles in programming practice. Proof construction is better suited to system specification and reasoning, whereas proof reduction is better suited to implementation.

To reduce the possibiity of error when building an implementation from a specification, we would like to minimize the gap between the two. Despite the apparent disparity between proof construction and proof reduction, is there a class of concurrent specifications from which decentralized concur-

rent implementations can be automatically extracted? Stated differently, is there perhaps (some fragment of) a substructural logic in which the computational natures of proof construction and proof reduction coincide?

1.1.2

The thesis is that, yes, we can indeed have our cake and eat it too:

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

Part I reviews some necessary background information: definitions of nondeterministic and deterministic finite automata (chapter 2), and a sequent calculus presentation of ordered logic (chapter 3). The reader should feel free to skim or skip these chapters and return to them

Part II then delves into a proof-construction approach to concurrency, beginning with a review of a framework for string rewriting (chapter 4). Because disjoint segments of a string may be rewritten independently, string rewriting can be used to specify concurrent systems that have a linear topology. The chapter concludes with two extended examples of string rewriting specifications: nondeterministic finite automata[20] and binary representations of natural numbers equipped with increment and decrement operations. These will serve as running examples throughout this document.

[20] And, as a special case, deterministic ones, too.

Despite being concurrent, string rewriting specifications lack an immediate notion of local execution. String rewriting presumes the existence of a central conductor that orchestrates the computation, rewriting the string globally. Global rewriting, although reasonable for concurrent specifications, will not map well to locally executing process implementations that a proof-reduction approach to concurrency suggests – the gap is simply too large.

Toward this end, chapter 5 presents an extension of the Lambek calculus[21] in which string rewriting specifications may be refined into *choreographies*. Inspired by the process-as-formula view of linear logic,[22] choreographies exist at a slightly lower level of abstraction than string rewriting specifications in that they begin to introduce a message-passing character. A choreography partitions a string rewriting specification's symbols into

[21] **Lambek:??**.

[22] **Miller:??**Cervesato and Scedrov 2009.

Having introduced a message-passing character and stronger notion of process identity, we can then ask when two processes are observationally equivalent. Chapter 7 develops a notion of bisimilarity for ordered propositions, where the uninterpreted atoms are the only observables.

1.1.3 *Contributions*

The contributions of this thesis can be viewed from several perspectives.

- This work can be seen as a proof-theoretic [logical] reconstruction of multiparty session types.[23] In multiparty session types, binary session types are generalized to conversations among several parties. Conversations in their entirety are specified using global session types. Global types can be projected to binary session types for each pair of participants, which very nearly are implementations.

[23] **Honda+:POPL08**.

- This work can be seen to further understanding of proof construction and proof reduction.

- Gives types to logic programs. Guarantees deadlock-freedom.

In addition to the practical benefit of

The remainder of this document aims to establish this thesis as a plausible one. To do so, we turn our attention from linear logic to (non-modal) intuitionistic ordered logic[24]—a restriction of linear logic in which the context of hypotheses forms a list rather than a multiset or bag—and defend the thesis in this restricted setting. The proposed thesis research is to relax the restrictions and expand the ideas in this document to intuitionistic linear logic.

[24] Lambek 1958; Polakow and Pfenning 1999.

Specifically, this document describes ... as depicted in **??**. First, **??** reviews a string rewriting interpretation of proof construction in a [non-modal] fragment of intuitionistic ordered logic.[25] String rewriting specifications in this fragment are equipped with a natural notion of concurrency based on treating as equivalent the different interleavings of independent rewriting steps. [equivalence classes of proofs.]

[25] **Simmons:CMU12**.

Despite being concurrent, these string rewriting specifications lack an immediate notion of *process* or *process identity*. Toward this end, **??** introduces *choreographies*, a further restriction of string rewriting specifications obtained when [in which] atomic propositions are assigned roles as either process-like atoms or message-like atoms. (Message-like atoms, such as in **??**, are indicated with an arrow decoration.) A specification may admit several choreographies, but, as described in **??**, a well-formed choreography must be (lock-step) equivalent with the specification.

Even with process-like atoms, choreographies remain string rewriting specifications, not actual distributed implementations of processes. Nevertheless, choreographies are a stepping-stone to process implementations. In **??**, we develop a session-typed process calculus from a Curry–Howard interpretation of a fragment of linear logic; **??** shows how choreographies can be compiled to

Choreographies serve as a stepping stone to full-fledged process definitions.

Part I

Preliminaries

2

Preliminaries: Automata

2.1 *Alphabets, words, and languages*

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

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^*.

It will sometimes be useful to work with an augmented alphabet. Given a symbol $c \notin \Sigma$, we may form the augmented alphabet $\Sigma_c = \Sigma \cup \{c\}$. The augmented alphabet Σ_ϵ is the most

2.1.1 *Endmarked alphabets and words*

2.2 *Nondeterministic finite automata*

Definition 2.1. An NFA over a finite alphabet Σ is a triple $\mathcal{A} = (Q, \longrightarrow, F)$ consisting of:

- a finite set of *states*, Q;
- a *transition relation*, $\longrightarrow \subseteq (Q \times \Sigma) \times Q$; and
- a subset of *final states*, $F \subseteq Q$.

We will write $q \xrightarrow{a} q'$ whenever $((q, a), q') \in \longrightarrow$.

The transition relation can be lifted to one involving finite input words: For each word $w = a_1 a_2 \cdots a_n \in \Sigma^*$, let $q \xrightarrow{w} q'$ if $q = q_0 \xrightarrow{a_1} q_1 \xrightarrow{a_2} \cdots \xrightarrow{a_n} q_n = q'$ 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 $\mathcal{L}_{\mathcal{A}}(q)$.[1]

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

Also, notice that, unlike most standard definitions 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.

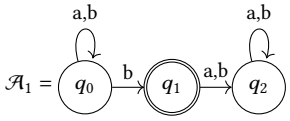


Figure 2.1: An NFA that accepts, from state q_0, exactly those words that end with b.

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. □

Definition 2.2. A deterministic finite automaton (DFA) is an NFA whose transition relation is, more precisely, a function.

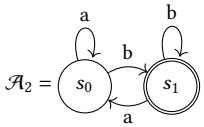


Figure 2.2: A DFA that accepts, from state s_0, exactly those words that end with b.

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. □

2.2.1 *NFA bisimilarity*

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

Input bisimilarity If $q \mathrel{\mathcal{R}} \xrightarrow{a} s'$, then $q \xrightarrow{a} \mathrel{\mathcal{R}} s'$.

Finality If $q \mathrel{\mathcal{R}} s \in F$, then $q \in F$.

NFA bisimilarity for $\mathcal{A}$, $\sim_{\mathcal{A}}$, is the largest bisimulation for $\mathcal{A}$.

Usually the automaton $\mathcal{A}$ is

Theorem 2.1. *Bisimilarity is an equivalence relation.*

Proof. Bisimilarity can be proved to be reflexive by showing that the state equality relation is a bisimulation and therefore included in the largest bisimulation. Bisimilarity is symmetric by definition. Bisimilarity can be proved to be transitive by showing that the relation $\sim\sim$ is a bisimulation. □

Theorem 2.2. *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be an NFA over the input alphabet Σ, and let $\mathcal{R}$ be a bisimulation for $\mathcal{A}$. 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. □

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

Proof. Because bisimilarity is symmetric, it suffices to show that $q \sim s$ implies $\mathcal{L}_{\mathcal{A}}(q) \subseteq \mathcal{L}_{\mathcal{A}}(s)$. Let q and s 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 **??** and [...] that $s \xrightarrow{w} s'_w \in F$, for some state s'_w. □

The converse is false.

FALSE CLAIM 2.4. *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be an NFA over the input alphabet Σ. If $\mathcal{L}_{\mathcal{A}}(q) = \mathcal{L}_{\mathcal{A}}(s)$, then $q \sim s$.*

Counterexample. Construct the NFA $\mathcal{A} = \mathcal{A}_1 \cup \mathcal{A}_2$ from the NFAs 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 $q_0 \sim s_0$ and its symmetric reflection, $s_0 \sim q_0$. Because q_0 is one of the b-successors of q_0, it follows by the input bisimilarity 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 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. □

2.2.2 *Nondeterministic finite automata with ϵ-transitions*

DEFINITION 2.4. An NFA with ϵ-moves over a finite alphabet Σ is a triple $\mathcal{A} = (Q, \longrightarrow, F)$ consisting of:

- a finite set of *states*, Q;
- a *transition relation*, $\longrightarrow \subseteq (Q \times \Sigma) \times Q$; and
- a subset of *final states*, $F \subseteq Q$.

We will write $q \overset{a}{\Longrightarrow} q'$ whenever $q \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} \xrightarrow{a} \xrightarrow{\epsilon} \cdots \xrightarrow{\epsilon} q'$.

The transition relation can be lifted to one involving finite input words: For each input word $w = a_1 a_2 \cdots a_n$, let $q \overset{w}{\Longrightarrow} q'$ if $q = q_0 \overset{a_1}{\Longrightarrow} q_1 \overset{a_2}{\Longrightarrow} \cdots \overset{a_n}{\Longrightarrow} q_n = q'$ for some sequence of states $q_0, q_1, \ldots, q_n \in Q$.

The automaton $\mathcal{A}$ accepts input word w from state q if there exists a state $q' \in Q$ such that $q \overset{w}{\Longrightarrow} q' \in F$; otherwise, the automaton rejects word w from state q.

2.3 *Finite transducers*

DEFINITION 2.5. A subsequential finite transducer (SFT) over a finite input alphabet Σ and a finite output alphabet Γ is a tuple $\mathcal{T} = (Q, \delta, \sigma, \rho)$ consisting of:

- a finite set of *states*, Q;
- a *transition function*, $\delta \colon Q \times \Sigma \to Q$;

- a *output function*, $\sigma\colon Q \times \Sigma \to \Gamma^*$; and
- a *terminal output function*, $\rho\colon Q \to \Gamma^*$.

Define a combined transition–output function $\longrightarrow\colon Q \times \Sigma \to \Gamma^* \times Q$:

$$q \xrightarrow{a|v} \delta(q,a), \text{ where } v = \sigma(q,a)$$

We then lift this function, defining a function $\stackrel{w}{\Longrightarrow}\colon Q \times \Sigma^* \to \Gamma^*$ on finite input words: For each input word $w = a_1\, a_2 \cdots a_n$:

$$q \stackrel{w}{\Longrightarrow} v$$

if and only if

$$q \xrightarrow{a_1|v_1} q_1 \xrightarrow{a_2|v_2} \cdots \xrightarrow{a_n|v_n} q_n \text{ and } v = (v_1\, v_2 \cdots v_n) \cdot \rho(q_n)\,.$$

The SFT $\mathcal{T}$ transforms, from state q, the input word w into the output word v if $(q,\epsilon) \stackrel{w}{\Longrightarrow} v$.

2.3.1 *From SFTs to proofs*

$$\Sigma^* \triangleq \oplus_{a\in\Sigma}\{a\colon \Sigma^*, \epsilon\colon \varepsilon\}$$
$$\Gamma^* \triangleq \oplus_{a\in\Gamma}\{a\colon \Gamma^*, \epsilon\colon \varepsilon\}$$
$$\Sigma^* \vdash \hat{q} : \Gamma^* \triangleq \mathsf{caseL}_{a\in\Sigma}(a \Rightarrow \hat{q}'_a \diamond \underset{\leftarrow}{\sigma}(q,a) \mid \epsilon \Rightarrow \underset{\rightarrow}{\rho}(q))$$

2.3.2 *From proofs to SFTs*

$$\Sigma^* \vdash p : \Gamma^* \triangleq \mathsf{caseL}_{a\in\Sigma}(a \Rightarrow p'_a \diamond \underset{\leftarrow}{v}_{p,a} \mid \epsilon \Rightarrow e_p \diamond \underset{\rightarrow}{v}'_{p,a})$$
$$\Sigma^* \vdash p : \Gamma^* \triangleq p' \diamond \underset{\leftarrow}{v}_p$$
$$\Sigma^* \vdash p : \varepsilon \triangleq \ldots$$
$$\varepsilon \vdash p : \Gamma^* \triangleq \ldots$$

2.4 *Deterministic pushdown automata*

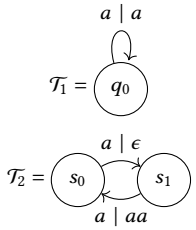


Figure 2.3: Two NFAs that copy streams of as.

Definition 2.6. A deterministic pushdown automaton (DPDA) over a finite input alphabet Σ and a finite stack alphabet Γ is a triple $\mathcal{A} = (Q, \longrightarrow, F)$ consisting of:

- a finite set of *states*, Q;
- a *transition relation* on state-stack pairs, $\longrightarrow \subseteq (Q \times \Gamma^*) \times \Sigma_\epsilon \times (Q \times \Gamma^*)$; and
- a subset of *final states*, $F \subseteq Q$.

We will write $q \xrightarrow{a} q'$ whenever $((q, a), q') \in \longrightarrow$.

The transition relation can be lifted to one involving finite input words: For each word $w = \alpha_1 \alpha_2 \cdots \alpha_n \in \Sigma^*$, let $q \xrightarrow{w} q'$ if $q = q_0 \xrightarrow{\alpha_1} q_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} q_n = q'$ for some sequence of states $q_0, q_1, \ldots, q_n \in Q$.

The DPDA $\mathcal{A}$ 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 $\mathcal{L}_{\mathcal{A}}(q)$.[2]

$(q, s) = (q_0, s_0) \xrightarrow{\alpha_1} (q_1, s_1) \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} (q', s')$ with $q' \in F$.

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

2.5 *Chains of communicating automata*

Definition 2.7. A *chain of communicating automata* over a finite alphabet Σ is a tuple $C = (Q, (\Sigma_q)_{q \in Q}, (\Gamma_q)_{q \in Q})$ consisting of:

- a finite set of *states*, Q, that is partitioned into: left- and right-reading states, Q^{rL} and Q^{rR}; left- and right-writing states, Q^{wL} and Q^{wR}; forking states, Q^{f}; and halting states, Q^{h};
- a state-indexed set of finite *left-hand alphabets*, $(\Sigma_q)_{q \in Q}$, and a state-indexed set of finite *right-hand alphabets*, $(\Gamma_q)_{q \in Q}$;
- $\delta^{\mathsf{rL}} \colon \Pi q{:}Q^{\mathsf{rL}}.\Sigma_q \to Q$, with the condition that $\Sigma_q = \Sigma_{q'}$ and $\Gamma_q = \Gamma_{q'}$ for all $q' \in \operatorname{cod} \delta^{\mathsf{rL}}_q$, and $\delta^{\mathsf{rR}} \colon (\exists q{:}Q^{\mathsf{rR}}.\Gamma_q) \to Q$;
- $\delta^{\mathsf{wL}} \colon Q^{\mathsf{wL}} \to \exists q'{:}Q.\Sigma_{q'}$ and $\delta^{\mathsf{wR}} \colon Q^{\mathsf{wR}} \to \exists q'{:}Q.\Gamma_{q'}$; and
- $\delta^{\mathsf{f}} \colon Q^{\mathsf{f}} \to Q \times Q$.

Chain configurations, c and d, consist of a finite sequence of states $q_1, q_2, \ldots, q_n \in Q$ with, for all $1 \leq i < n$, a finite word drawn from $\Sigma^*_{q_{i+1}}$ between neighboring states q_i and q_{i+1}. In addition, a finite word drawn from $\Sigma^*_{q_1}$ and a finite word drawn from $\Gamma^*_{q_n}$ bracket the configuration. Formally, then, a chain configuration is a string drawn from the set

$$\Sigma^*_{q_1} q_1 \Sigma^*_{q_2} q_2 \cdots \Sigma^*_{q_n} q_n \Gamma^*_{q_n},$$

for some finite sequence of states $q_1, q_2, \ldots, q_n \in Q$. The chain configuration is *well-formed* if $\Gamma_{q_i} = \Sigma_{q_{i+1}}$ for all $1 \leq i < n$.

- $c\, a\, q\, d \longrightarrow c\, q'\, d$ if $\delta^{\mathsf{rL}}_q(a) = q'$, and $c\, q\, a\, d \longrightarrow c\, q'\, d$ if $\delta^{\mathsf{rR}}_q(a) = q'$;
- $c\, q\, d \longrightarrow c\, a\, q'\, d$ if $\delta^{\mathsf{wL}}(q) = (q', a)$, and $c\, q\, d \longrightarrow c\, q'\, a\, d$ if $\delta^{\mathsf{wR}}(q) = (q', a)$;
- $c\, q\, d \longrightarrow c\, q'\, q''\, d$ if $\delta^{\mathsf{f}}(q) = (q', q'')$;
- $c\, q\, d \longrightarrow c\, d$ if $q \in Q^{\mathsf{h}}$ and $\Sigma_q = \Gamma_q$.

3
Preliminaries: 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] Ibid.

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.

This chapter serves to review a sequent calculus presentation of ordered logic. Lambek originally developed the calculus 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.

As a substructural logic, ordered logic eschews the usual structural properties of antecedents – weakening, contraction, and exchange. As in Girard's linear logic,[5] the lack of weakening and contraction properties means that each antecedent must be used exactly once within a proof. Ordered logic's additional lack of an exchange property means that antecedents must also remain in order within a proof.

[5] Girard 1987.

> Lambek leveraged this 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 Pfenning (2016) and Polakow and Pfenning (1999).

Section 3.1 introduces ordered logic's sequent calculus as a collection of in-

ference rules, informally justifying them with a resource interpretation similar to that of linear logic.[6] To be sure that this collection ... Together, cut and identity elimination theorems (theorems 3.2 and 3.4) serve to establish a proof normalization result: every proof has a corresponding verification.

[6] ??.

The reader who is familiar with ordered logic's sequent calculus and 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*, 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 $A_2 \cdots A_n \vdash A$ nor $A_1\, A_1\, A_2 \cdots A_n \vdash A$ implies $A_1\, A_2 \cdots A_n \vdash A$, 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 $\Omega \vdash A$. Algebraically, ordered contexts Ω form a free noncommutative monoid:

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

where the monoid operation is concatenation, denoted by juxtaposition, and the unit element is the empty context, denoted by $(\cdot)$. As a monoid, ordered

contexts are equivalent up to associativity and unit laws (see adjacent figure). We choose to keep this equivalence implicit, however, treating equivalent contexts as syntactically indistinguishable. Associativity means that contexts are indeed lists, not trees; and noncommutativity means that those lists are ordered, not multisets as in linear logic.

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

Figure 3.1: The monoid laws for ordered contexts

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, C should be able to be produced 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 **??**, the admissibility of these principles serves an important role in defining the meaning of the logical connectives.

3.1.3 *The logical connectives*

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

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

Among these are propositional atoms, α, 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[8] 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[9], so $\mathbf{1}$ is indeed $\bullet$'s unit.

DISJUNCTION AND ITS UNIT Disjunction is the proposition $A \oplus B$, read "A plus B".[10] 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.

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.

[8] Also known as multiplicative conjunction.

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

[10] 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).

ALTERNATIVE CONJUNCTION AND ITS UNIT Alternative conjunction[11] 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.

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

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, $\&\text{R}$, 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, $\setminus\text{L}$, 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, $\setminus\text{R}$, 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 $A \setminus (B \setminus C) \dashv\vdash (B \bullet A) \setminus C$ and $(C \,/\, B) \,/\, A \dashv\vdash C \,/\, (A \bullet B)$ are derivable.

[11] Also known as additive conjunction.

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

The previous section presented a collection of inference rules that have a[n apparently] sensible resource interpretation. But how can we be sure that the rules constitute a well-defined *logic* and not merely a ...?

Ordered conjunction	$A \bullet B$	A side-by-side pair of resources A and B, packaged as a single resource
(Additive) disjunction	$A \oplus B$	A resource package that contains A or B (but not both)
Alternative conjunction	$A \mathbin{\&} B$	A resource that can be transformed into either A or B
Left-handed implication	$A \setminus B$	A resource that transforms a left-adjacent resource A into resource B
Right-handed implication	$A \setminus B$	A resource that transforms a right-adjacent resource A into resource B

Table 3.1: A reminder of ordered logic's connectives

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

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

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

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

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

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

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

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

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

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

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

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

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.[12] 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[13] propositions. [In this way, the meaning of a proposition is compositional.]

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

[13] lemmas?

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

The end result will be a cut-free, 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 a* cut-free *proof of* $\Omega \vdash A$ *exists.*

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

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

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 a* cut-free *proof of* $\Omega'_L\, \Omega\, \Omega'_R \vdash C$ *exists.*

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

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

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.

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 a* cut-free *proof of* $\Omega'_L\, \Omega\, \Omega'_R \vdash C$ *exists.*

Proof. This lemma was proved in a similar setting by Polakow and Pfenning (1999) using a standard technique for proving the admissibility of a cut principle[14] – 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.

[14] Pfenning 1995.

As usual, the various cases can be sorted into three classes: identity cut reductions, principal cut reductions, and commutative cut reductions.

Identity cut reductions 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:

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

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

Principal cut reductions 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:

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

$$=$$

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

Commutative cut reductions 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 cut reductions 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:

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

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 a* cut-free *proof of* $\Omega \vdash A$ *exists.*

Proof. We follow the proof sketched by Polakow and Pfenning (ibid.). 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.

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[15] proofs to cut-free form.

[15] Correct word?

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 (page 35) that the next step toward proof normalization is to eliminate all remaining instances of the ID rule that occur at compound propositions A. Before proving the identity elimination **??**, we need to prove that an identity principle is admissible for long proofs.

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

Proof. As usual[16], by induction on the structure of the proposition A. As before, we may represent this lemma as an admissible rule:

[16] Best reference? Frank's lecture notes?

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

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

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

$$\dfrac{}{\alpha \vdash^{\ell} \alpha}\ \textsc{a-id}^{\alpha} \quad = \quad \dfrac{}{\alpha \vdash^{\ell} \alpha}\ \textsc{id}^{\alpha}$$

For compound propositions, the 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:

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

The remaining cases are similarly compositional. □

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

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

$$\dfrac{\Omega \vdash A}{\Omega \vdash^{\ell} A}\ \textsc{ie}$$

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 a long proof of $\Omega \vdash A$:

$$\dfrac{\dfrac{}{A \vdash A}\ \textsc{id}^{A}}{A \vdash^{\ell} A}\ \textsc{ie} \quad = \quad A \vdash^{\ell} A\ \textsc{a-id}^{A}$$

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

The remaining cases are resolved compositionally. For example:

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

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(theorem 3.2 and **??**) 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, long proof) of* $\Omega \vdash A$ *exists.*

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

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

□

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 *Extensions*

In this section, we give a brief overview of several extensions to the preceding ordered sequent calculus: first-order universal and existential quantifiers, multiplicative falsehood, and mobility and persistence modalities. These extensions are not crucial to the remainder of this dissertation, but are included for the sake of completeness.

FIRST-ORDER QUANTIFICATION 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, Σ, of well-sorted term variables, $x{:}\tau$; this new context is structural, admitting weakening, contraction, and exchange properties.

$$\frac{\Sigma, a{:}\tau; \Omega \vdash [a/x]A}{\Sigma; \Omega \vdash \forall x{:}\tau.A}\ \forall\text{R} \qquad \frac{\Sigma \vdash t : \tau \quad \Sigma; \Omega'_L\ ([t/x]A)\ \Omega'_R \vdash C}{\Sigma; \Omega'_L\ (\forall x{:}\tau.A)\ \Omega'_R \vdash C}\ \forall\text{L}$$

$$\frac{\Sigma \vdash t : \tau \quad \Sigma; \Omega \vdash [t/x]A}{\Sigma; \Omega \vdash \exists x{:}\tau.A}\ \exists\text{R} \qquad \frac{\Sigma, a{:}\tau; \Omega'_L\ ([a/x]A)\ \Omega'_R \vdash C}{\Sigma; \Omega'_L\ (\exists x{:}\tau.A)\ \Omega'_R \vdash C}\ \exists\text{L}$$

MULTIPLICATIVE FALSEHOOD Along a different dimension, the ordered sequent calculus can be generalized to allow sequents to carry an empty consequent, $\Omega \vdash \cdot$. With this new judgment form, the cut principle and left rules must be revised to allow the empty consequent. For example, the CUT and $\bullet$L rules are revised to:

$$\frac{\Omega \vdash A \quad \Omega'_L\ A\ \Omega'_R \vdash \gamma}{\Omega'_L\ \Omega\ \Omega'_R \vdash \gamma}\ \text{CUT}^A \qquad \frac{\Omega'_L\ A\ B\ \Omega'_R \vdash \gamma}{\Omega'_L\ (A \bullet B)\ \Omega'_R \vdash \gamma}\ \bullet\text{L}$$

where γ is a metavariable standing for a consequent, either empty, $\cdot$, or a single proposition, C.

$\gamma ::= \cdot \mid C$

This new judgment makes it possible to introduce *multiplicative falsehood*, $\bot$, as a logical constant. Multiplicative falsehood is, as its name suggests, dual to multiplicative truth, $\mathbf{1}$. Its right and left rules are:

$$\frac{\Omega \vdash \cdot}{\Omega \vdash \bot}\ \bot\text{R} \qquad \frac{}{\bot \vdash \cdot}\ \bot\text{L}$$

MOBILITY AND PERSISTENCE MODALITIES Linear[17]

[17] Circular proofs, too!

3.4 *Circular propositions and circular derivations*

- No exponentials; recursion/circularity instead (Milner)[18]
- μMALL (Baelde) and circular proofs (Fortier and Santocanale)
- Contractivity requirement

[18] Should this go in ordered rewriting chapter instead?

- We will use only general recursion. Inductive and coinductive types are outside our scope.
- Subset of infinite propositions/derivations

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 behavior[1] of concurrent systems. This is not, of course, a new idea. Multiset rewriting,[2] and other state-transformation models such as Petri nets,[3] [...][4] Unlike the application domain for multiset rewriting, we are particularly interested in concurrent systems whose components are arranged in a linear topology and have a monoidal structure. Given that finite strings over an alphabet Σ form a monoid, string rewriting is a good match for this structure.

[1] operational semantics?
[2] ??.
[3] ??.
[4] fix

For a broad sketch of string rewriting, consider the finite strings over the alphabet $\{a, b\}$, and let $\longrightarrow$ be the least compositional[5] binary relation over those strings that satisfies the axioms

[5] compatible?

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

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 operation 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 the choices. [6]

[6] Bring up concurrency here?

The remainder of this chapter describes the 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 *String rewriting*

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.[7] 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.

[7] As usual for a free monoid, the alternative grammar, $w ::= \epsilon \mid a\, w$, describes the same strings.

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

Figure 4.1: The 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 compositional[8] relation satisfying a collection of rewriting axioms, chosen on a per-application basis, such as the axioms

$$\frac{}{a\, b \longrightarrow b} \qquad \text{and} \qquad \frac{}{b \longrightarrow \epsilon} \tag{4.2}$$

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 forbidden.

[8] compatible?

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 $\longrightarrow_\Theta$.

In addition to the application-specific axioms contained within the signature, rewriting is always permitted within substrings, so we adopt two compatibility rules. Together, the rules

$$\frac{w_1 \longrightarrow w_1'}{w_1\, w_2 \longrightarrow w_1'\, w_2}\ \longrightarrow_{\mathrm{C_L}} \qquad \text{and} \qquad \frac{w_2 \longrightarrow w_2'}{w_1\, w_2 \longrightarrow w_1\, w_2'}\ \longrightarrow_{\mathrm{C_R}}.$$

ensure that the rewriting relation is compatible with the [free][9] monoidal structure of strings.[10]

The $\longrightarrow$ relation 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 $\longrightarrow$.[11]

Consistent with its [free][12] 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,

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

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. 4.2.

4.1.3 *Properties of string rewriting*

As an abstract rewriting system, string rewriting can be evaluated for several [standard][13] 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 (??). 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.

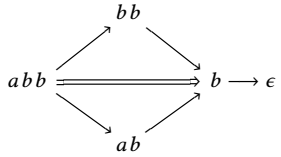


Figure 4.3: An example of concurrent string rewriting

[9] fix

[10] Because strings form a monoid, these compatibility rules are equivalent to the unified rule

$$\frac{w \longrightarrow w'}{w_L\, w\, w_R \longrightarrow w_L\, w'\, w_R}\ \longrightarrow_{\mathrm{C}}.$$

However, we prefer the two-rule formulation because it aligns more closely with the free monoid's syntactic structure.

[11] Usually written as $\longrightarrow^*$, we instead chose $\Longrightarrow$ 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 clear in subsequent chapters.

[12] fix

[13] fix

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

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

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

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

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

Figure 4.2: A string rewriting framework

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 the first rewriting step occurs. [14]

More generally, this idea that the interleaving of independent actions is irrelevant is known as *concurrent equality*,[15] and it forms the basis of concurrency.[16] With the partial commutativity endowed by concurrent equality, the [free] monoid formed by rewriting sequences is, more specifically, a trace monoid. As such, we will frequently refer to rewriting sequences as *traces*.

[14] explain figure

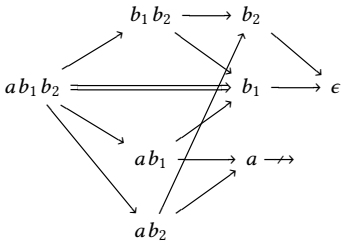


Figure 4.4: When multiple occurences of b are properly distinguished, a complete trace diagram can be given.

[15] **Watkins+:CMU02.**

[16] **??.**

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'.

Because string rewriting is an asymmetric, committed-choice relation, 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. 4.4. 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, however, 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

$$\overline{a \longrightarrow b} \qquad \text{and} \qquad \overline{b \longrightarrow a}\,.$$

Every finite trace from a nonempty string can always be extended by applying one of these axioms, so rewriting never terminates.

4.2 *Extended 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 $\mathcal{A} = (Q, ?, F)$[17] 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[18] ϵ to act as an end-of-word marker, along with axioms[19]

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

These axioms imply that rewriting occurs over the strings $\{\epsilon\} \times \Sigma^* \times Q$.

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

$$\overline{a\, q_0 \longrightarrow q_0} \qquad \overline{b\, q_0 \longrightarrow q_0} \text{ and } \overline{b\, q_0 \longrightarrow q_1} \qquad \overline{\epsilon\, q_0 \longrightarrow n}$$

$$\overline{a\, q_1 \longrightarrow q_2} \qquad \overline{b\, q_1 \longrightarrow q_2} \qquad \overline{\epsilon\, q_1 \longrightarrow \cdot}$$

$$\overline{a\, q_2 \longrightarrow q_2} \qquad \overline{b\, q_2 \longrightarrow q_2} \qquad \overline{\epsilon\, q_2 \longrightarrow n}$$

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

$$\epsilon\, b\, b\, a\, q_0 \longrightarrow \epsilon\, b\, b\, q_0 \longrightarrow \epsilon\, b\, q_0 \longrightarrow \epsilon\, q_1 \longrightarrow (\cdot)\,.$$

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.

THEOREM 4.2 (Adequacy of NFA specification). *Let $\mathcal{A} = (Q, ?, F)$*[20] *be an NFA over the input alphabet Σ.*

[17] fix

[18] replace with $?

[19] Check these with choreography

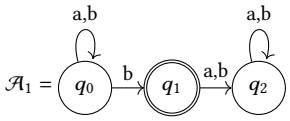


Figure 4.5: An NFA that accepts, from state q_0, exactly those words that end with b. (Repeated from fig. 2.1.)

$$(w_1\, w_2)^{\mathrm{R}} = w_2^{\mathrm{R}}\, w_1^{\mathrm{R}}$$
$$\epsilon^{\mathrm{R}} = \cdot$$
$$a^{\mathrm{R}} = a$$

Figure 4.6: An (anti-)homomorphism for reversal of finite words

[20] fix

- $q \xrightarrow{a} q'_a$ *if, and only if,* $a\, q \longrightarrow q'_a$*, for all input symbols* $a \in \Sigma$*.*
- $q \in F$ *if, and only if,* $\epsilon\, q \longrightarrow (\cdot)$*.*
- $q \xrightarrow{w} q'$ *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 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 *Extended 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, or *binary counters*. 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 $\Omega = e\, b_1$ and $\Omega' = e\, b_0\, b_1$ 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 $\Omega \approx_{\mathrm{v}} n$, we say that Ω denotes, or represents, natural number n in binary.

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

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 Ω that form the relation's domain of definition.[21]

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

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

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

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

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

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

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

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

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

Proof. The three claims may be proved by induction over the structure of Ω, 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 $\approx_v$ relation non-injective: for example, both $e\,b_1 \approx_v 1$ and $e\,b_0\,b_1 \approx_v 1$ hold. However, if the rule for b_0 is restricted to *nonzero* even numbers, then each natural number has a unique, canonical representation that is free of leading b_0s.[22]

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

$$\frac{\Omega \approx_v n \quad (n > 0)}{\Omega\,b_0 \approx_v 2n}\,.$$

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: $\Omega \equiv \Omega'$ if $\Omega \approx_v n$ and $\Omega' \approx_v n$ for some n.

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 Ω that represents n, we may append i to form an active[23], computational string, $\Omega\,i$. For i to adequately represent the increment operation, the string $\Omega\,i$ must meet two conditions, captured by the following global desiderata:

- $\Omega\,i \Longrightarrow\!\approx_v n+1$ – that is, *some* rewriting sequence results in a binary representation of $n+1$; and
- $\Omega\,i \Longrightarrow \Omega'$ implies $\Omega' \Longrightarrow\!\approx_v n+1$ – that is, *any* rewriting sequence from $\Omega\,i$ can[24] result in a binary representation of $n+1$.

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

[23] The 'active', 'latent', and 'passive' terminology is borrowed from **Pfenning+Simmons:LICS09**. 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.

[24] ??

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:

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

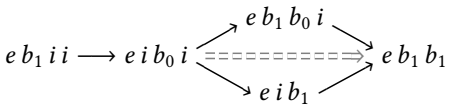
These three axioms can be read as follows:

- To increment e, append b_1 as a new most[25] significant bit, resulting in $e\,b_1$.
- 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.

[25] or least?

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.[26]

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

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 $\Omega \approx_{\text{V}} n$, then $\Omega \approx_{\text{I}} n$ and $\Omega \not\longrightarrow$.*
Preservation *If $\Omega \approx_{\text{I}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\text{I}} n$.*
Progress *If $\Omega \approx_{\text{I}} n$, then either: $\Omega \longrightarrow \Omega'$ for some Ω'; or; $\Omega \approx_{\text{V}} n$.*[27]
Termination *If $\Omega \approx_{\text{I}} n$, then every rewriting sequence from Ω is finite.*

Proof. Each part is proved separately.

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

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

Termination can be proved using an explicit termination measure, $|-|_{\text{I}}$, that is strictly decreasing across each rewriting, $\Omega \longrightarrow \Omega'$. Specifically, we use a measure (see the adjacent figure), adapted from the standard amortized constant work analysis of increment for binary counters.[28] The measure $|-|_{\text{I}}$ is such that $\Omega \longrightarrow \Omega'$ implies $|\Omega|_{\text{I}} > |\Omega'|_{\text{I}}$; because the measure is always nonnegative, only finitely many such rewritings can occur.

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

Corollary 4.5 (Big-step adequacy of increments).
Evaluation *If $\Omega \approx_{\text{I}} n$, then $\Omega \Longrightarrow \approx_{\text{V}} n$. In particular, if $\Omega \approx_{\text{V}} n$, then $\Omega\ i \Longrightarrow \approx_{\text{V}} n+1$.*
Preservation *If $\Omega \approx_{\text{I}} n$ and $\Omega \Longrightarrow \Omega'$, then $\Omega' \approx_{\text{I}} n$. In particular, if $\Omega \approx_{\text{V}} n$ and $\Omega\ i \Longrightarrow \Omega'$, then $\Omega' \Longrightarrow \approx_{\text{V}} n+1$.*

[26] 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:

$$\Omega ::= e \mid \Omega\ b_0 \mid \Omega\ b_1 \mid \Omega\ i$$

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

[27] Compare with "If $\Omega \approx_{\text{I}} n$, then $\Omega \approx_{\text{V}} n$ if, and only if, $\Omega \not\longrightarrow$."

$$|e|_{\text{I}} = 0$$
$$|\Omega\ b_0|_{\text{I}} = |\Omega|_{\text{I}}$$
$$|\Omega\ b_1|_{\text{I}} = |\Omega|_{\text{I}} + 1$$
$$|\Omega\ i|_{\text{I}} = |\Omega|_{\text{I}} + 2$$

Figure 4.7: A termination measure, adapted from the standard amortized work analysis of increment for binary counters

[28] ??.

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' interchangably.) A string Ω will be said to be in head-unary form if it has one of two forms: $\Omega = z$; or $\Omega = \Omega' s$, for some binary number Ω'.

Just as appending the symbol i to a counter Ω 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 Ωd must satisfy the following global desiderata when $\Omega \approx_{\mathrm{D}} n$:

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

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

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 $\Omega\, b_1$ denotes $2n+1$ if Ω denotes n, its head-unary form, $\Omega\, b_0\, s$, can be constructed by flipping the least significant bit to b_0 and appending s.
- Because $\Omega\, b_0$ denotes $2n$ if Ω denotes n, its head-unary form can be contructed by recursively putting the more significant bits, Ω, into head-unary form and appending b_0' to process that result.

- If Ω has head-unary form z and therefore denotes 0, then $\Omega\, b_0$ also denotes 0 and has head-unary form z.
- Otherwise, if Ω has head-unary form $\Omega'\, s$ and thus denotes $n > 0$, then $\Omega\, b_0$ denotes $2n > 0$ and has head-unary form $\Omega'\, b_1\, s$, 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 $e\, b_0\, b_1\, s$:

$$e\, b_1\, i\, d \longrightarrow e\, i\, b_0\, d \;\Rightarrow\; \begin{matrix} e\, b_1\, b_0\, d \\ \\ e\, i\, d\, b_0' \end{matrix} \;\Rightarrow\; e\, b_1\, d\, b_0' \longrightarrow e\, b_0\, s\, b_0' \longrightarrow e\, b_0\, b_1\, s\ .$$

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, $\approx_{\text{D}}$, that assigns a natural number denotation to each string that may appear as an intermediate state during a decrement.

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

At first glance, the d-D rule may look a bit odd: Why is the denotation unchanged by a decrement, $\Omega\, d$? 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.[29]

Also, notice that the premises of the d-D and s-D rules use the increment-only denotation relation, $\approx_{\text{I}}$, not the decrement relation, $\approx_{\text{D}}$. 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 $\approx_{\text{D}}$ relation, so d may have b_0' symbols to its right.

With this $\approx_{\text{D}}$ 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 $\Omega \approx_{\text{D}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\text{D}} n$.*
Progress *If $\Omega \approx_{\text{D}} n$, then [either]:*[30]

- $\Omega \longrightarrow \Omega'$, *for some* Ω';
- $n = 0$ *and* $\Omega = z$; *or*
- $n > 0$ *and* $\Omega = \Omega'\, s$, *for some* Ω' *such that* $\Omega' \approx_{\text{I}} n - 1$.

Termination *If $\Omega \approx_{\text{D}} n$, then every rewriting sequence from Ω is finite.*

Proof. Each part is proved separately.

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

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

$$\Omega ::= e \mid \Omega\, b_0 \mid \Omega\, b_1 \mid \Omega\, i$$
$$\Delta ::= \Omega\, d \mid \Delta\, b_0' \mid z \mid \Omega\, s$$
$$[\![\Omega\, d]\!]_{\text{D}} = [\![\Omega]\!]_{\text{I}}$$
$$[\![\Delta\, b_0']\!]_{\text{D}} = 2[\![\Delta]\!]_{\text{D}}$$
$$[\![z]\!]_{\text{D}} = 0$$
$$[\![\Omega\, s]\!]_{\text{D}} = [\![\Omega]\!]_{\text{I}} + 1$$

[30] ?

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.[31]

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

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

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

Figure 4.8: A termination measure for decrements, where $|\Omega|$ denotes the length of string Ω

[31] 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 document, so we use the simpler linear potential.

Corollary 4.7 (Big-step adequacy of decrements). *If* $\Omega \approx_{\mathrm{D}} n$*, then:*

- $\Omega \Longrightarrow z$ *if, and only if,* $n = 0$*;*
- $\Omega \Longrightarrow \Omega'\, s$ *for some* Ω' *such that* $\Omega' \approx_{\mathrm{I}} n - 1$*, if* $n > 0$*; and*
- $\Omega \Longrightarrow \Omega'\, s$ *only if* $n > 0$ *and* $\Omega' \approx_{\mathrm{I}} n - 1$*.*

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 $\Omega \approx_{\mathrm{D}} n$ and $\Omega \Longrightarrow \Omega'$ only if $\Omega' \approx_{\mathrm{D}} n$. 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

In this chapter, we develop a rewriting interpretation of the ordered sequent calculus from the previous chapter.

In 1958, Lambek developed a syntactic calculus, now known as the Lambek calculus, for formally describing the structure of sentences.[1] Words are assigned syntactic types, which roughly correspond to grammatical parts of speech. From a logical perspective, the Lambek calculus can [also] be viewed as a precursor to (and generalization of) Girard's linear logic[2].[3] Implicit in Lambek's original article is a third perspective of the calculus: string rewriting.

In this chapter, we review the Lambek calculus from a [string] rewriting perspective.

[1] Lambek 1958.

[2] Girard 1987.

[3] Polakow and Pfenning 1999**Polakow+Pfenning:TLCA99**.

The previous chapter showed how to use string rewriting to specify, on a global level, the [...] of concurrent systems that have a linear topology. Although useful for [...], these string rewriting specifications lack a clear notion of local, decentralized execution – for each step of rewriting, the entire string is rewritten as a monolithic whole by a central conductor.

Keeping in mind our ultimate goal of decentralized[4] implementations of concurrent systems, these string rewriting specifications are too abstract. Instead, we need to expose local interactions that are left implicit in the string rewriting specifications.

[4] distributed?

As an example, recall from chapter 4 the string rewriting specification of a system that may transform strings that end with b into the empty string:

$$\overline{a\,b \longrightarrow b} \qquad \overline{b \longrightarrow \cdot}\,. \tag{5.1}$$

This specification is non-local in two ways: the central conductor must identify those substrings that can be rewritten according to one of the axioms. In the [...] axiom, for example, there is no description of how the symbols a and b would identify each other and coordinate to effect a rewriting to b.

To [...], we introduce *choreographies*, which refine string rewriting specifications by consistently assigning each symbol one of two roles: message or

process.

$$\frac{}{\underset{\rightarrow}{a}\,\hat{b} \longrightarrow \hat{b}} \qquad \text{and} \qquad \frac{}{\hat{b} \longrightarrow \cdot}$$

a recursively defined ordered proposition, such as

$$\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \,\&\, \mathbf{1}$$

for the process $\hat{b}$.

The remainder of this chapter presents a formulation of the Lambek calculus from the ordered sequent calculus of chapter 3.

Then, in

One valid choreography for this specification views each symbol b as a process that nondeterministically receives some number of messages a before terminating.

If we annotate messages with an underbar and processes with a circumflex, then $\underset{\rightarrow}{a}\,\hat{b} \longrightarrow \hat{b}$ and $\hat{b} \longrightarrow \cdot$.

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:

$$\frac{\Omega'_L\, A\, B\, \Omega'_R \vdash C}{\Omega'_L\, (A \bullet B)\, \Omega'_R \vdash C}\ \bullet\text{L} \qquad \text{and} \qquad \frac{\Omega'_L\, A\, \Omega'_R \vdash C}{\Omega'_L\, (A \,\&\, B)\, \Omega'_R \vdash C}\ \&\text{L}_1 .$$

Both rules decompose the principal resource: in the $\bullet$L rule, $A \bullet B$ into the separate resources $A\,B$; and, in the $\&$L$_1$ rule, $A \,\&\, B$ 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 $\bullet$L and $\&$L$_1$ rules, but are general boilerplate that arguably should be factored out.

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 a generalization of the string rewriting shown in chapter 4.

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

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

The standard left rules can then be recovered from resource decomposition rules using this cut principle. 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 $\bullet$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

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

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

$$\frac{\Omega \vdash A}{\Omega\,(A \setminus B) \longrightarrow B} \ \setminus\text{D}' \qquad \text{and} \qquad \frac{\Omega \vdash A}{(B \mathbin{/} A)\,\Omega \longrightarrow B} \ /\text{D}' . \tag{5.2}$$

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

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 $\setminus$D$'$ and /D$'$ rules have premises, and those premises create a dependence of the decomposition judgment upon general provability. Second, the above $\setminus$D$'$ and /D$'$ rules do not decompose the principal proposition into *immediate* subformulas since Ω is involved. This contrasts with, for example, the $\bullet$D rule that decomposes $A \bullet B$ into the immediate subformulas $A\,B$.

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

$$\frac{}{A\,(A \setminus B) \longrightarrow B} \ \setminus\text{D} \qquad \text{and} \qquad \frac{}{(B \mathbin{/} A)\,A \longrightarrow B} \ /\text{D},$$

which we adopt as decomposition rules in place of those in eq. (5.2). The standard $\setminus$L and /L rules can still be recovered from these more specific decomposition rules, thanks to 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.[5]

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, $\oplus$L, doesn't decompose the resource $A \oplus B$. The $\oplus$L rule certainly does decompose $A \oplus B$, but it does so [...].[6] $A \oplus B \longrightarrow A \mid B$ [...] retain the standard $\oplus$L and $\mathbf{0}$L rules.

FIGURE 5.21 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).

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

$$\leftrightsquigarrow$$

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

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

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

$$\leftrightsquigarrow$$

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

Figure 5.2: A possible refactoring of the $\setminus$L rule in terms of resource decomposition

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

$$\leftrightsquigarrow$$

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

Figure 5.3: Refactoring the $\setminus$L rule in terms of resource decomposition, via $\setminus$D and CUT$^{\longrightarrow}$

[5] Lambek 1958.

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

[6] fix

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

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

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

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

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

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

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

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

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

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

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

THEOREM 5.1 (Soundness). *If $\Omega \vdash A$ is derivable in the refactored calculus of fig. 5.21, then $\Omega \vdash A$ is derivable in the usual ordered sequent calculus (fig. 3.2).*

Proof. By structural induction on the given derivation. The key lemma is the admissibility of $\text{CUT}^{\longrightarrow}$ in the 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. 5.18 and 5.20. □

THEOREM 5.2 (Completeness). *If $\Omega \vdash A$ is derivable in the usual ordered sequent calculus (fig. 3.2), then $\Omega \vdash A$ is derivable in the refactored calculus of fig. 5.21.*

Proof. 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. 5.18 and 5.20. □

5.1.2 *Ordered resource decomposition as rewriting*

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. Instead, we can also 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 $\bullet\text{D}$ 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'."

Figure 5.22 summarizes the rewriting system that we obtain from the refactored sequent-like calculus of fig. 5.21. Essentially, the ordered rewriting system 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 $\backslash\text{D}$ 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. 5.21, this kind of framing is taken care of by the cut principle for decomposition, $\text{CUT}^{\longrightarrow}$. To express framing at the level of the $\Omega \longrightarrow \Omega'$ judgment itself, we introduce two compatibility rules: together,

$$\frac{\Omega_1 \longrightarrow \Omega'_1}{\Omega_1\ \Omega_2 \longrightarrow \Omega'_1\ \Omega_2}\ {\longrightarrow}\text{C}_\text{L} \qquad \text{and} \qquad \frac{\Omega_2 \longrightarrow \Omega'_2}{\Omega_1\ \Omega_2 \longrightarrow \Omega_1\ \Omega'_2}\ {\longrightarrow}\text{C}_\text{R}$$

ensure that rewriting is compatible with concatenation of ordered contexts.[7]

By forming the reflexive, transitive closure of $\longrightarrow$, we may construct a multi-step rewriting relation, which we choose to write as $\Longrightarrow$.[8]

Consistent with its [free] monoidal structure, there are two equivalent formulations of this reflexive, transitive closure: each rewriting sequence $\Omega \Longrightarrow \Omega'$ can be viewed as either a list or tree of individual rewriting steps.[9]

[7] Because ordered contexts form a monoid, these compatibility rules are equivalent to the unified rule

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

However, we prefer the two-rule formulation of compatibility because it better aligns with the syntactic structure of contexts.

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

[9] rewrite with reference to string rewriting

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

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

$$\frac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\text{D} \qquad \frac{}{(B \,/\, A)\,A \longrightarrow B}\ /\text{D}$$

(no $\oplus$D and $\mathbf{0}$D rules)

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1'\,\Omega_2}\ \longrightarrow\text{C}_\text{L} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1\,\Omega_2'}\ \longrightarrow\text{C}_\text{R}$$

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

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

We prefer the list-based formulation shown in fig. 5.22 because it tends to [...] proofs by structural induction, but, on the basis of the following **??**, we allow ourselves to freely switch between the two formulations as needed.

FACT 5.3 (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 $\setminus$D and /D rules. The $\setminus$D′ and /D′ rules, with their $\Omega \vdash A$ premises, would be problematic as rewriting rules because they would introduce a dependence of ordered rewriting upon general provabilityand the accompanying proof search would take ordered rewriting too far afield from traditional, syntactic[10] notions of string and multiset rewriting.

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

FALSE CLAIM 5.4 (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).

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. □

[10] Is this the right word? [mechanical, computational]

$$(\Omega_1\,\Omega_2) = (\Omega_1) \bullet (\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$A = A$$

$$\bullet(\Omega_1\,\Omega_2) = (\bullet\Omega_1) \bullet (\bullet\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$\bullet A = A$$

Figure 5.6: From ordered contexts to propositions

Last, 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.[11] We may then prove the following lemma.

[11] use a or p in place of α?

$$
\begin{aligned}
|\Omega_1\ \Omega_2|_\star &= |\Omega_1|_\star + |\Omega_2|_\star \\
|\cdot|_\star &= 0 \\
|A \star B|_\star &= 1 + |A|_\star + |B|_\star \\
&\qquad \text{if } \star = \bullet, \mathbin{\&}, \setminus, /, \text{ or } \oplus \\
|A|_\star &= 1 \quad \text{if } A = \alpha, \mathbf{1}, \top, \text{ or } \mathbf{0}
\end{aligned}
$$

Figure 5.7: A measure of the number of logical connectives within an ordered context

Lemma 5.6. *If $\Omega \longrightarrow \Omega'$, then $|\Omega|_\star > |\Omega'|_\star$. Also, if $\Omega \Longrightarrow \Omega'$, then $|\Omega|_\star \geq |\Omega'|_\star$.*

Proof. By induction on the structure of the rewriting step. □

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

Theorem 5.7 (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 5.6 establishes $|\Omega_{i+1}|_\star < 0$, which is impossible because $|-|_\star$ is a measure. □

5.1.3 *Concurrency in ordered rewriting*

As an example of multi-step rewriting, observe that

$$a_1\ (a_1 \setminus a_2)\ (b_2 \mathbin{/} b_1)\ b_1 \Longrightarrow a_2\ b_2.$$

In fact, as shown in the adjacent figure, two sequences witness this rewriting: either the initial state's left half, $a_1\ (a_1 \setminus a_2)$, is first rewritten to a_2 and then its right half, $(b_2 \mathbin{/} b_1)\ b_1$, is rewritten to b_2; or *vice versa*, the right half is first rewritten to b_2 and then the left half is rewritten to a_2.

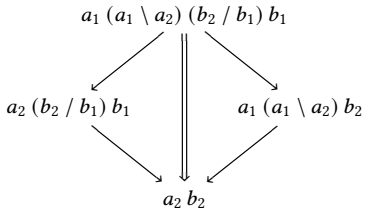


Figure 5.8: An example of concurrent ordered rewriting

Notice that these two sequences differ only in how non-overlapping, and therefore independent, rewritings of the initial state's two halves are interleaved. Consequently, 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 $a_1\ (a_1 \setminus a_2)\ (b_2 \mathbin{/} b_1)\ b_1$ to $a_2\ b_2$ remains.

More generally, this idea that the interleaving of independent actions is irrelevant is known as *concurrent equality*,[12] and it forms the basis of concurrency.[13] Concurrent equality also endows traces $\Omega \Longrightarrow \Omega'$ with a free partially commutative monoid structure, *i.e.*, traces form a trace monoid.

[12] **Watkins+:CMU02.**

[13] ??.

5.1.4 *Other properties of ordered rewriting*

As the relation $\Longrightarrow$ forms a rewriting system, we may evaluate it along several standard dimensions: termination, confluence.

Although terminating, ordered rewriting is not confluent. Confluence requires that all states with a common ancestor, *i.e.*, states Ω_1' and Ω_2' such that $\Omega_1' \Longleftarrow\!\!\Longrightarrow \Omega_2'$, be joinable, *i.e.*, $\Omega_1' \Longrightarrow\Longleftarrow \Omega_2'$. Because ordered rewriting is directional[14] and the relation $\Longrightarrow$ is not symmetric, some nondeterministic choices are irreversible.

[14] Is this phrasing correct?

FALSE CLAIM 5.8 (Confluence). *If* $\Omega_1' \Longleftarrow\!\!\Longrightarrow \Omega_2'$, *then* $\Omega_1' \Longrightarrow\Longleftarrow \Omega_2'$.

Counterexamples. Consider the state $\alpha \,\&\, \beta$. By the rewriting rules for additive conjunction, $\alpha \longleftarrow \alpha \,\&\, \beta \longrightarrow \beta$, and hence $\alpha \Longleftarrow \alpha \,\&\, \beta \Longrightarrow \beta$. However, being atoms, neither α nor β reduces. And $\alpha \neq \beta$, so $\alpha \Longrightarrow\Longleftarrow \beta$ does *not* hold.

Even in the $\&$-free fragment, ordered rewriting is not confluent. For example,

$$\not\longleftarrow b_1\,(a \backslash b_2) \Longleftarrow (b_1 \,/\, a)\, a\, (a \backslash b_2) \Longrightarrow (b_1 \,/\, a)\, b_2 \not\longrightarrow .$$ □

5.2 *Unbounded ordered rewriting*

[15]

[15] Aranda et al. 2007.

Although a seemingly pleasant property, termination (??) significantly limits the expressiveness of ordered rewriting. For example, without unbounded rewriting, we cannot even give ordered rewriting specifications of producer-consumer systems or finite automata.

As the proof of termination shows, rewriting is bounded precisely because states consist of finitely many finite propositions. To admit unbounded rewriting, we therefore choose to permit infinite propositions in the form of mutually recursive definitions, $\hat{p} \triangleq A$. These definitions are collected into a signature, $\Sigma = (\hat{p}_i \triangleq A_i)_i$, which indexes the rewriting relations: $\longrightarrow_\Sigma$ and $\Longrightarrow_\Sigma$.[16] To rule out definitions like $\hat{p} \triangleq \hat{p}$ that do not correspond to sensible infinite propositions, we also require that definitions be *contractive*[17] – *i.e.*, that the body of each recursive definition begin with a logical connective at the top level.

[16] We frequently elide the indexing signature, as it is usually clear from context.

[17] Gay and Hole 2005.

By analogy with recursive types from functional programming,[18] we must now decide whether to treat definitions *iso*recursively or *equi*recursively. Under an equirecursive interpretation, definitions $\hat{p} \triangleq A$ may be silently unrolled or rolled at will; in other words, $\hat{p}$ is literally *equal* to its unrolling, A. In contrast, under an isorecursive interpretation, unrolling a recursively defined proposition would count as an explicit step of rewriting – $\hat{p} \longrightarrow A$, for example.

[18] ??.

We choose to interpret definitions equirecursively because the equirecursive treatment, with its generous notion of equality, helps to minimize the overhead of recursively defined propositions. As a simple example, under the equirecursive definition $\hat{p} \triangleq a \backslash \hat{p}$, we have the trace

$$a\, a\, \hat{p} = a\, a\, (a \backslash \hat{p}) \longrightarrow a\, \hat{p} = a\, (a \backslash \hat{p}) \longrightarrow \hat{p}$$

or, more concisely, $a\,a\,\hat{p} \longrightarrow a\,\hat{p} \longrightarrow \hat{p}$. Had we chosen an isorecursive treatment of the same definition, we would have only the more laborious

$$a\,a\,\hat{p} \longrightarrow a\,a\,(a \setminus \hat{p}) \longrightarrow a\,\hat{p} \longrightarrow a\,(a \setminus \hat{p}) \longrightarrow \hat{p}.$$

PROPOSITIONS $A ::= a \mid \hat{p} \mid A \bullet B \mid \mathbf{1} \mid A \setminus B \mid B \,/\, A \mid A \,\&\, B \mid \top$

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

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

$$\frac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\mathrm{D} \qquad \frac{}{(B \,/\, A)\,A \longrightarrow B}\ /\mathrm{D}$$

(no $\oplus$D and $\mathbf{0}$D rules)

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1'\,\Omega_2}\longrightarrow\mathrm{C_L} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1\,\Omega_2'}\longrightarrow\mathrm{C_R}$$

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

5.2.1 *Replication*

In Milner's development of the π-calculus, there are two avenues to unbounded process behavior: recursive process definitions and replication.

5.3 *Focused ordered rewriting*

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 $a\,\big((a \setminus b) \,\&\, \top\big) \longrightarrow a\,(a \setminus b)$. It is not possible to rewrite $a\,\big((a \setminus b) \,\&\, \top\big)$ into b in a single step, although it is possible in several steps: $a\,\big((a \setminus b) \,\&\, \top\big) \Longrightarrow b$, because $a\,\big((a \setminus b) \,\&\, \top\big) \longrightarrow a\,(a \setminus b) \longrightarrow b$.

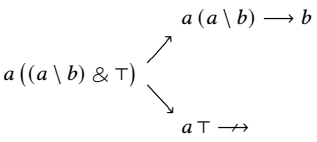


Figure 5.9:

The decomposition rules are so fine-grained that rewriting may get stuck in many undesirable ways. For instance, in the previous example, we might have instead nondeterministically committed to rewriting $a\,\big((a \setminus b) \,\&\, \top\big)$ into $a\,\top$ as the first step, and then $a\,\top$ is stuck, with no further rewritings possible.

Chaining and inversion as coarse-grained decomposition: all the way to a shift or atom.[19]

The ordered propositions are polarized into positive and negative classes, or *polarities*[20], according to the invertibility of their sequent calculus rules;

[19] origin of focused (multiset) rewriting?

[20] reference?

two 'shift' connectives, $\downarrow$ and $\uparrow$, mediate between the two classes.

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

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

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

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

$$\Omega^\mp ::= \Omega_1^\mp \; \Omega_2^\mp \mid \cdot \mid A^- \mid a^+$$

As usual, we do not distinguish those ordered contexts that are equivalent up to the monoid laws.

Each class of propositions is equipped with its own focusing judgment: a *left-focus judgment*, $\Omega_L^\mp \; [A^-] \; \Omega_R^\mp \Vdash C^+$, that focuses on a negative proposition, A^-, that occurs to the left of the turnstile; and a *right-focus judgment*, $[A^+] \dashv\Vdash \Omega^\mp$, that focuses on a positive proposition, A^+, that occurs to the right of the turnstile.[21]

[21] We choose to write the turnstile [...].

Following **Zeilberger:??**, each of these judgments can be read as [describing] 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, $[A^+] \dashv\Vdash \Omega^\mp$, decomposes A^+ into an ordered context $\Omega^\mp$ of its [nearest] negative subformulas, and is given by the following rules.

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

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

Ordered conjunctions $A^+ \bullet B^+$ are decomposed into $\Omega_1^\mp \; \Omega_2^\mp$ by inductively decomposing A^+ and B^+ into $\Omega_1^\mp$ and $\Omega_2^\mp$, respectively, and $\mathbf{1}$ is decomposed into the empty context. Atoms a^+ cannot be decomposed further[22], and $\downarrow A^-$ is decomposed into its immediate subformula of negative polarity, A^-.

[22] Alternatively, a^+ could be decomposed to a suspension, $\langle a^+ \rangle$.

LEMMA 5.9. *$[\bullet\Omega_2^\mp] \dashv\Vdash \Omega_1^\mp$ if, and only if, $\Omega_1^\mp = \Omega_2^\mp$.*

Proof. Each direction is separately proved by structural induction on the context $\Omega_2^\mp$. □

$$\bullet(\Omega_1^\mp \; \Omega_2^\mp) = (\bullet\Omega_1^\mp) \bullet (\bullet\Omega_2^\mp)$$

$$\bullet(\cdot) = \mathbf{1}$$

$$\bullet A^- = \downarrow A^-$$

$$\bullet a^+ = a^+$$

Figure 5.10: From ordered contexts to propositions

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^+, and is given by the following rules.

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

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

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

Unlike the right-focus judgment, this left-focus judgment describes a nondeterministic function (or relation).

These rules parallel the usual sequent calculus rules, maintaining focus on the immediate subformulas. The

$$\frac{\Omega_L\ [A^-]\ \Omega_R \Vdash B^+ \quad [B^+] \dashv \Omega'}{\Omega_L\ A^-\ \Omega_R \longrightarrow \Omega'}\ ?$$

In addition, the compatibility rules $\longrightarrow\text{C}_\text{L}$ and $\longrightarrow\text{C}_\text{R}$ are retained:

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\ \Omega_2 \longrightarrow \Omega_1'\ \Omega_2}\ \longrightarrow\text{C}_\text{L} \qquad \text{and} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1\ \Omega_2 \longrightarrow \Omega_1\ \Omega_2'}\ \longrightarrow\text{C}_\text{R}\,.$$

5.3.1 *Minimal polarization*

Focused ordered rewriting is sound with respect to the unfocused rewriting framework of **??**. Given a depolarization function

$$(A)^{\boxplus} = \begin{cases} a^+ & \text{if } A = a \\ A_1^{\boxplus} \bullet A_2^{\boxplus} & \text{if } A = A_1 \bullet A_2 \\ \mathbf{1} & \text{if } A = \mathbf{1} \\ \downarrow A^{\boxminus} & \text{otherwise} \end{cases}$$

$$(A)^{\boxminus} = \begin{cases} A_1^{\boxplus} \setminus A_2^{\boxminus} & \text{if } A = A_1 \setminus A_2 \\ A_2^{\boxminus} / A_1^{\boxplus} & \text{if } A = A_2 / A_1 \\ A_1^{\boxminus} \mathbin{\&} A_2^{\boxminus} & \text{if } A = A_1 \mathbin{\&} A_2 \\ \top & \text{if } A = \top \\ \uparrow A^{\boxplus} & \text{otherwise} \end{cases}$$

$$\Omega^{\square} = \begin{cases} \Omega_1^{\square}\ \Omega_2^{\square} & \text{if } \Omega = \Omega_1\ \Omega_2 \\ \cdot & \text{if } \Omega = \cdot \\ A^{\boxminus} & \text{if } \Omega = A \neq a \\ a^+ & \text{if } \Omega = a \end{cases}$$

THEOREM 5.10. *If* $\Omega_1^{\square} \longrightarrow \Omega_2^{\mp}$, *then* $\Omega_1 \Longrightarrow \Omega_2$ *for some* Ω_2 *such that* $\Omega_2^{\square} = \Omega_2^{\mp}$.

Figure 5.11: Focused ordered rewriting

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

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

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

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

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

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

$$\frac{\Omega_L \; [A^-] \; \Omega_R \Vdash B^+ \quad [B^+] \dashv\Vdash \Omega'}{\Omega_L \; A^- \; \Omega_R \longrightarrow \Omega'}$$

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1 \; \Omega_2 \longrightarrow \Omega_1' \; \Omega_2}\;\longrightarrow\text{C}_\text{L} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1 \; \Omega_2 \longrightarrow \Omega_1 \; \Omega_2'}\;\longrightarrow\text{C}_\text{R}$$

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

5.3.2 *Embedding unfocused ordered rewriting*

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

- $\Omega \longrightarrow \Omega'$ implies $\Omega^{\square} \longrightarrow (\Omega')^{\square}$; and
- $\Omega^{\square} \longrightarrow \Delta'$ implies $\Omega \longrightarrow \Omega'$, for some Ω' such that $\Delta' = (\Omega')^{\square}$.[23]

[23] That is, $(-)^{\square}$ is a strong reduction bisimulation**Sangiorgi+Walker:CUP03**.

By appropriately placing double shifts, $\downarrow\uparrow$ or $\uparrow\downarrow$, between each pair of

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 interupted after each step, forcing the focused rewriting to mimic the small-step behavior of unfocused rewriting.

The mapping $(-)^{\square}$ relies on three auxiliary mappings: $(-)^{\square}$, from unpolarized propositions to negative propositions and positive atoms; $()^{\boxminus}$, from unplarized propositions to negative propositions; and $()^{\boxplus}$, from unpolarized propositions to

$A^{\boxminus}$ and $A^{\boxplus}$ produce negative and positive polrizations of A, respectively, that insert a $\downarrow\uparrow$ shift in front of every proper nonatomic subformula of A. In addition, $A^{\boxminus}$ prepends an $\uparrow$ shift when the top-level connective of A has positive polarity, whereas $A^{\boxminus}$ prepends a $\downarrow$ shift when A is not atomic. $A^{\square}$ produces either a positive atom or a negative polarization of A, according to wheter A is atomic.

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

Figure 5.12: An embedding of unfocused ordered rewriting within the focused ordered rewriting framework

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

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

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

$$[A^{\boxplus}] \dashv\Vdash \Omega \text{ if, and only if, } \Omega = A^{\square}.$$

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 $\Omega = A_1\ (A_1 \setminus A_2) \longrightarrow A_2 = \Omega'$. By definition, $\Omega^{\square} = A_1^{\square}\ (A_1^{\boxplus} \setminus \uparrow A_2^{\boxplus})$ and $(\Omega')^{\square} = A_2^{\square}$, and we may derive $A_1^{\square}\ [A_1^{\boxplus} \setminus \uparrow A_2^{\boxplus}] \Vdash A_2^{\boxplus}$ and $[A_2^{\boxplus}] \dashv\Vdash A_2^{\square}$. So, indeed, $\Omega^{\square} = A_1^{\square}\ (A_1^{\boxplus} \setminus \uparrow A_2^{\boxplus}) \longrightarrow A_2^{\square} = (\Omega')^{\square}$.

The second property is also proved by induction over the structure of the given rewriting step, this time $\Omega^{\square} = \Delta \longrightarrow \Delta'$. As an example, consider the case in which $\Omega_L^{\square}\ [A_1^{\boxplus} \setminus \uparrow A_2^{\boxplus}]\ \Omega_R^{\square} \Vdash B^{+}$ and $[B^{+}] \dashv\Vdash \Delta'$, for some Ω_L, A_1, A_2, and Ω_R such that $\Omega = \Omega_L\ (A_1 \setminus A_2)\ \Omega_R$. By inversion, $\Omega_L = A_1$, $\Omega_R = \cdot$, $B^{+} = A_2^{\boxplus}$, and $\Delta' = A_2^{\square}$. Indeed, $\Omega = A_1\ (A_1 \setminus A_2) \longrightarrow A_2 = \Omega'$ and $\Delta' = (\Omega')^{\square}$. □

5.3.3 *Embedding weakly focused ordered rewriting*

It is similarly possible to embed weakly focused ordered rewriting, a rewriting discipline that lies between unfocused ordered rewriting and fully focused ordered rewriting. In weakly focused rewriting, chaining of negative propositions is retained, but positive propositions are not eagerly inverted like they are in fully focused rewriting. For example,

$$a^+ \downarrow \hat{p}^- = a^+ \downarrow \big((a^+ \setminus \uparrow(\downarrow \hat{p}^- \bullet a^+)) \mathbin{\&} (b^+ \setminus \uparrow \mathbf{1})\big) \longrightarrow \downarrow \hat{p}^- \bullet a^+ \longrightarrow \downarrow \hat{p}^- \; a^+$$

This weak focusing discipline can be achieved

$$\frac{\Omega_L^+ \; [A^-] \; \Omega_R^+ \Vdash C^+}{\Omega_L^+ \downarrow A^- \; \Omega_R^+ \longrightarrow C^+} \downarrow_{\mathrm{D}} \qquad \frac{}{A^+ \bullet B^+ \longrightarrow A^+ \; B^+} \bullet_{\mathrm{D}} \qquad \frac{}{\mathbf{1} \longrightarrow \cdot} \mathbf{1}_{\mathrm{D}}$$

$$\frac{\Omega_1^+ \longrightarrow \Omega_1^+}{\Omega_1^+ \; \Omega_2^+ \longrightarrow \Omega_1^+ \; \Omega_2^+} \longrightarrow_{\mathrm{C_L}} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1 \; \Omega_2 \longrightarrow \Omega_1 \; \Omega_2'} \longrightarrow_{\mathrm{C_R}}$$

$$\frac{\Omega_L^+ \; [B^-] \; \Omega_R^+ \Vdash C^+}{\Omega_L^+ \; A^+ \; [A^+ \setminus B^-] \; \Omega_R^+ \Vdash C^+} \setminus_{\mathrm{L}}' \qquad \frac{\Omega_L \; [B^-] \; \Omega_R \Vdash C^+}{\Omega_L \; [B^- \,/\, A^+] \; A^+ \; \Omega_R \Vdash C^+} /_{\mathrm{L}}'$$

$$(A^+)^{\boxplus} = \begin{cases} a^+ & \text{if } A^+ = a^+ \\ \downarrow\uparrow((A_1^+)^{\boxplus} \bullet (A_2^+)^{\boxplus}) & \text{if } A^+ = A_1^+ \bullet A_2^+ \\ \downarrow\uparrow \mathbf{1} & \text{if } A^+ = \mathbf{1} \\ \downarrow (A_0^-)^{\boxminus} & \text{if } A^+ = \downarrow A_0^- \end{cases}$$

$$(A^-)^{\boxminus} = \begin{cases} (A_1^+)^{\boxplus} \setminus (A_2^-)^{\boxminus} & \text{if } A^- = A_1^+ \setminus A_2^- \\ (A_2^-)^{\boxminus} \,/\, (A_1^+)^{\boxplus} & \text{if } A^- = A_2^- \,/\, A_1^+ \\ (A_1^-)^{\boxminus} \mathbin{\&} (A_2^-)^{\boxminus} & \text{if } A^- = A_1^- \mathbin{\&} A_2^- \\ \top & \text{if } A^- = \top \\ \uparrow (A_0^+)^{\boxplus} & \text{if } A^- = \uparrow A_0^+ \end{cases}$$

$$(A^+)^{\square} = \begin{cases} a^+ & \text{if } A^+ = a^+ \\ \uparrow((A_1^+)^{\boxplus} \bullet (A_2^+)^{\boxplus}) & \text{if } A^+ = A_1^+ \bullet A_2^+ \\ \uparrow \mathbf{1} & \text{if } A^+ = \mathbf{1} \\ (A_0^-)^{\boxminus} & \text{if } A^+ = \downarrow A_0^- \end{cases}$$

$$(\Omega^+)^{\square} = \begin{cases} (\Omega_1^+)^{\square} \; (\Omega_2^+)^{\square} & \text{if } \Omega^+ = \Omega_1^+ \; \Omega_2^+ \\ (\cdot) & \text{if } \Omega = \cdot \\ (A^+)^{\square} & \text{if } \Omega = A^+ \end{cases}$$

THEOREM 5.12. *The embedding* $(-)^{\square}$ *satisfies the following properties.*

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

Proof. The first property is proved by induction over the structure of the given weakly focused rewriting step, $\Omega^+ \longrightarrow \Omega'^+$, with the inductive hypothesis generalized to include

- If $\Omega_L^+ \, [A^-] \, \Omega_R^+ \Vdash C^+$, then $(\Omega_L^+)^\square \, [(A^-)^\boxminus] \, (\Omega_R^+)^\square \Vdash (C^+)^\boxplus$.

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 inductive hypothesis, $(\Omega_L^+)^\square \, [(A^-)^\boxminus] \, (\Omega_R^+)^\square \Vdash (C^+)^\boxplus$ and so $(\Omega_L^+)^\square \, (\downarrow A^-)^\square \, (\Omega_R^+)^\square \longrightarrow (C^+)^\square$.

- If $\Omega_L^- \, [(A^-)^\boxminus] \, \Omega_R^- \Vdash B^+$, then $\Omega_L^- = (\Omega_L^+)^\square$ and $\Omega_R^- = (\Omega_R^+)^\square$ and $B^+ = (C^+)^\boxplus$ and $\Omega_L^+ \, [A^-] \, \Omega_R^+ \Vdash C^+$.

The second property is also proved by induction over the structure of the given rewriting step, this time the fully focused $(\Omega^+)^\square \longrightarrow \Omega^-{}'$. As an example, consider the case in which $\Omega_L^- \, [(A_1^+)^\boxplus \setminus (A_2^-)^\boxminus] \, \Omega_R^- \Vdash B^+$ and $[(B^+)^\boxplus] \dashv \Omega^-{}'$. By the inductive hypothesis, $\Omega_L^+ \, [A_2^-] \, \Omega_R^+ \Vdash C^+$ such that $\Omega_L^- = (\Omega_L^+)^\square$, $\Omega_R^- = (\Omega_R^+)^\square$, and $B^+ = (C^+)^\boxplus$. It follows that $\Omega_L^+ \, A_1^+ \, [A_1^+ \setminus A_2^-] \, \Omega_R^+ \Vdash C^+$, and so $\Omega_L^+ \downarrow (A_1^+ \setminus A_2^-) \, \Omega_R^+ \longrightarrow ?$ □

5.3.4 *Recursively defined propositions and focused ordered rewriting*

Consider the recursively defined proposition $p \triangleq (a \backslash p \bullet a) \mathbin{\&} (b \backslash \mathbf{1})$. Previously, in the unfocused rewriting framework, it took three steps to rewrite $a\, p$ into $p\, a$:

$$a\, p = a \, \big((a \setminus p \bullet a) \mathbin{\&} (b \setminus \mathbf{1})\big) \longrightarrow a \, (a \setminus p \bullet a) \longrightarrow p \bullet a \longrightarrow p \, a \, .$$

Now, in the polarized, focused rewriting framework, the analogous recursive definition with only the necessary shifts is $\hat{p}^- \triangleq (a^+ \setminus \uparrow(\downarrow \hat{p}^- \bullet a^+)) \mathbin{\&} (b^+ \setminus \uparrow \mathbf{1})$, and it takes only one focused step to rewrite $a^+ \, \hat{p}^-$ into $\hat{p}^- \, a^+$:

$$a^+ \, \hat{p}^- = a^+ \, \big((a^+ \setminus \uparrow(\downarrow \hat{p}^- \bullet a^+)) \mathbin{\&} (b^+ \setminus \uparrow \mathbf{1})\big) \longrightarrow \hat{p}^- \, a^+$$

because

$$a^+ \, [\hat{p}^-] \Vdash \downarrow \hat{p}^- \bullet a^+ \qquad \text{and} \qquad [\downarrow \hat{p}^- \bullet a^+] \dashv \hat{p}^- \, a^+ \, .$$

Because the left-focus judgment is defined inductively, not coinductively, there are some recursively defined negative propositions that cannot successfully be put in focus. For example, under the definition $\hat{p}^- \triangleq a^+ \setminus \hat{p}^-$, there are no contexts Ω_L and Ω_R and positive consequent C^+ for which $\Omega_L \, [\hat{p}^-] \, \Omega_R \Vdash C^+$ is derivable. To derive a left-focus judgment on $\hat{p}^-$, the finite context Ω_L would need to hold an infinite stream of a^+ atoms, which is impossible in an inductively defined context.

However, by inserting a double shift, $\uparrow\downarrow$, which allows focus to be blurred at the $\uparrow$, the definition can be revised to one that admits left-focusing: when $\hat{p}^-$ is defined by $\hat{p}^- \triangleq a^+ \setminus \uparrow\downarrow \hat{p}^-$, the judgment $a^+ \, [\hat{p}^-] \Vdash \downarrow \hat{p}^-$ is derivable. It follows that $a^+ \, \hat{p}^- \longrightarrow \hat{p}^-$.

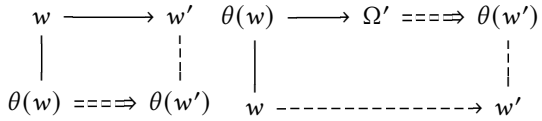
More generally, any recursively defined proposition that does not pass through an $\uparrow$ shift along a main branch cannot be successfully put in focus.

5.4 *Choreographies*

Recall the string rewriting specification

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

A choreography is a refinement of this specification in which each symbol a of the rewriting alphabet is mapped to an ordered proposition: either an atomic proposition, $\underset{\leftarrow}{a}$ or $\underset{\rightarrow}{a}$, or a recursively defined proposition, $\hat{a}$. In other words, a choreography is an injection from symbols to propositions.



$\underset{\rightarrow}{a}\,\hat{b}$

Suppose that θ is the mapping $a \mapsto \underset{\rightarrow}{a}$ and $b \mapsto \hat{b}$. and the choreography

$$\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \hat{b}) \,\&\, \mathbf{1}\,.$$

Notice that

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

as well as

$$\underset{\rightarrow}{a}\,\hat{b} \longrightarrow \underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \hat{b}) \longrightarrow \hat{b} \quad \text{and} \quad \hat{b} \longrightarrow \mathbf{1} \longrightarrow \cdot\,.$$

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

$$\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \hat{b}) \,\&\, (\underset{\rightarrow}{c} \setminus \hat{b})$$

$$a\,b \longrightarrow w' \text{ implies } w' = b \quad \text{but} \quad \underset{\rightarrow}{a}\,\hat{b} \longrightarrow \underset{\rightarrow}{a}\,(\underset{\rightarrow}{c} \setminus \hat{b}) \not\longrightarrow$$

[24]

[24] McDowell+:TCSo3.

Judgments $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ and $\theta \vdash w \longrightarrow w'\,[\hat{a}]\,A \rightsquigarrow$. In both judgments, all terms before the $\rightsquigarrow$ are inputs; all terms after the $\rightsquigarrow$ are outputs.

$$\frac{}{\theta \vdash \cdot \rightsquigarrow \cdot} \qquad \frac{\theta \vdash \Sigma \rightsquigarrow \Sigma' \quad \theta \vdash w \longrightarrow w'\,[\hat{a}]\,A_2 \rightsquigarrow \quad (\Sigma'(\hat{a}) = A_1)}{\theta \vdash \Sigma, w \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq A_1 \,\&\, A_2}$$

$$\frac{\theta \vdash \Sigma \rightsquigarrow \Sigma' \quad \theta \vdash w \longrightarrow w'\,[\hat{a}]\,A \rightsquigarrow \quad (\hat{a} \notin \operatorname{dom}\Sigma')}{\theta \vdash \Sigma, w \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq A}$$

$$\frac{(\theta(a) = \hat{a}) \quad (\theta(w') = \Omega')}{\theta \vdash a \longrightarrow w'\,[\hat{a}]\uparrow(\bullet\Omega') \rightsquigarrow}$$

$$\frac{\theta \vdash w \longrightarrow w'\,[\hat{a}]\,A \rightsquigarrow \quad (\theta(b) = \underset{\rightarrow}{b})}{\theta \vdash b\,w \longrightarrow w'\,[\hat{a}]\,\underset{\rightarrow}{b} \setminus A \rightsquigarrow} \qquad \frac{\theta \vdash w \longrightarrow w'\,[\hat{a}]\,A \rightsquigarrow \quad (\theta(b) = \underset{\leftarrow}{b})}{\theta \vdash w\,b \longrightarrow w'\,[\hat{a}]\,A \,/\, \underset{\leftarrow}{b} \rightsquigarrow}$$

THEOREM 5.13. • *If* $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ *and* $w \longrightarrow_{\Sigma} w'$, *then* $\theta(w) \longrightarrow_{\Sigma'} \theta(w')$. *If* $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ *and* $\Omega \longrightarrow_{\Sigma'} \Omega'$, *then* $\theta^{-1}(\Omega) \longrightarrow_{\Sigma} \theta^{-1}(\Omega')$.

• *If* $\theta \vdash w \longrightarrow w' \, [\hat{a}] \, A \rightsquigarrow$, *then* $\theta(w) \longrightarrow_{\hat{a} \triangleq A} \theta(w')$. *If* $\theta \vdash w \longrightarrow w' \, [\hat{a}] \, A \rightsquigarrow$ *and* $\Omega \longrightarrow_{\hat{a} \triangleq A} \Omega'$, *then* $\theta^{-1}(\Omega) \longrightarrow \theta^{-1}(\Omega')$.

Proof. $\hat{a} \longrightarrow \bullet \theta(w')$
$\underset{\rightarrow}{b} \, \theta(w) \longrightarrow_{\hat{a} \triangleq \underset{\rightarrow}{b} \backslash A} \theta(w')$ if $\theta(w) \longrightarrow_{\hat{a} \triangleq A} \theta(w')$
$\theta(w) \, \underset{\leftarrow}{b} \longrightarrow_{\hat{a} \triangleq A / \underset{\leftarrow}{b}} \theta(w')$ if $\theta(w) \longrightarrow_{\hat{a} \triangleq A} \theta(w')$

□

When $\theta = \{(a, \underset{\rightarrow}{a}), (b, \hat{b})\}$, the judgment $\theta \vdash \Sigma \rightsquigarrow \hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \hat{b}) \,\&\, \mathbf{1}$ holds. However, $b \longrightarrow \epsilon$ but $\hat{b} \not\longrightarrow \cdot$.

$$\begin{aligned} \hat{e} &\triangleq \uparrow(\downarrow \hat{e} \bullet \downarrow \hat{b}_1) \,/\, \underset{\leftarrow}{i} \\ \hat{b}_0 &\triangleq \uparrow \downarrow \hat{b}_1 \,/\, \underset{\leftarrow}{i} \\ \hat{b}_1 &\triangleq \uparrow(\underset{\leftarrow}{i} \bullet \downarrow \hat{b}_0) \,/\, \underset{\leftarrow}{i} \end{aligned}$$

$$\hat{e} \triangleq (\uparrow(\downarrow \hat{e} \bullet \downarrow \hat{b}_1) \,/\, \underset{\leftarrow}{i}) \,\&\, (\uparrow \underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d})$$

$$\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \uparrow(\underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1)) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \uparrow \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \uparrow(\downarrow \hat{\imath} \bullet \underset{\rightarrow}{b}_0))$$

5.4.1

5.5

Atomic ordered propositions are viewed as messages; compound ordered propositions, as processes; and ordered contexts, as configurations of processes.

The ordered contexts form a monoid over the positive propositions and are given by

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

In keeping with the monoid laws, we treat $(\Omega_1 \, \Omega_2) \, \Omega_3$ and $\Omega_1 \, (\Omega_2 \, \Omega_3)$ as syntactically indistinguishable, as we also do for $\Omega \, (\cdot)$ and Ω and $(\cdot) \, \Omega$.

Each atom is consistently assigned a direction, either left-directed, $\underset{\leftarrow}{a}$, or right-directed, $\underset{\rightarrow}{a}$.

An atom's direction and position within the larger context together indicate whether, when viewed as a message, it is being sent or received. In the context $\Omega_1 \, \underset{\rightarrow}{a} \, \Omega_2$, the atom $\underset{\rightarrow}{a}$ is a message being sent from Ω_1 to Ω_2. Symmetrically, in the context $\Omega_1 \, \underset{\leftarrow}{a} \, \Omega_2$, the atom $\underset{\leftarrow}{a}$ is a message being sent from Ω_1 to Ω_2.

The context $\Omega = \Omega' \, \underset{\rightarrow}{a}$ is a process configuration that sends $\underset{\rightarrow}{a}$ to its right and continues as Ω'. Conversely, $\underset{\rightarrow}{a} \, \Omega$ is a process configuration in which Ω is the intended recipient of the message $\underset{\rightarrow}{a}$.

$$A^+ ::= \underset{\leftarrow}{a} \mid \underset{\rightarrow}{a} \mid \hat{p}^+ \mid A^+ \bullet B^+ \mid \mathbf{1} \mid \downarrow A^-$$
$$A^- ::= \hat{p}^- \mid \underset{\rightarrow}{a} \setminus B^- \mid B^- \,/\, \underset{\leftarrow}{a} \mid A^- \,\&\, B^- \mid \top \mid \uparrow A^+$$

5.6 *Choreographing specifications*

$$\overline{a\, b \longrightarrow b} \qquad \text{and} \qquad \overline{b \longrightarrow \cdot}$$

As a specification, these string rewriting axioms are quite reasonable. However, as a [...], [...].

Toward our ultimate goal of relating the proof-construction and proof-reduction appraches to concurrency, we would like a description of this concurrent system that is slightly more concrete.

$$\underset{\rightarrow}{a}\, \hat{b} \longrightarrow \hat{b}$$
$$\hat{b} \longrightarrow \cdot$$

$$\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow \hat{b}) \,\&\, \mathbf{1}$$

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

$$\hat{e} \triangleq \hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}$$
$$\hat{b}_0 \triangleq \hat{b}_1 \,/\, \underset{\leftarrow}{i}$$
$$\hat{b}_1 \triangleq \underset{\leftarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\leftarrow}{i}$$

$$e \mathrel{\mathcal{R}} \hat{e}$$
$$b_0 \mathrel{\mathcal{R}} \hat{b}_0$$
$$b_1 \mathrel{\mathcal{R}} \hat{b}_1$$
$$i \mathrel{\mathcal{R}} \underset{\leftarrow}{i}$$
$$e\, b_1 \mathrel{\mathcal{R}} \hat{e} \bullet \hat{b}_1$$
$$i\, b_0 \mathrel{\mathcal{R}} \underset{\leftarrow}{i} \bullet \hat{b}_0$$

$\mathcal{R}$ is a reduction bisimulation.

$\hat{e}\, \underset{\leftarrow}{i} \Longrightarrow \hat{e}\, \hat{b}_1$ and $e\, i \Longrightarrow e\, b_1$ and $e\, b_1 \Longrightarrow e\, b_1$

$$\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b_1}) \,\&\, (\underset{\rightarrow}{b_0} \setminus \underset{\rightarrow}{b_1}) \,\&\, (\underset{\rightarrow}{b_1} \setminus \hat{\imath} \bullet \underset{\rightarrow}{b_0})$$

$\underset{\rightarrow}{e}\, \hat{\imath} \Longrightarrow \underset{\rightarrow}{e}\, \underset{\rightarrow}{b_1}$ and $e\, i \Longrightarrow e\, b_1$ and $e\, b_1 \Longrightarrow e\, b_1$

$$\hat{q} \triangleq \mathop{\&}_{a \in \Sigma} (\underset{\rightarrow}{a} \setminus \hat{q}'_a)$$

Compare:

- $q \xrightarrow{a} q'_a$ if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a$.
- $q \xrightarrow{a} q'_a$ if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$ for some $\Omega' = \hat{q}'_a$.
- $q \xrightarrow{a} q'_a$ and $\hat{q}'_a = \Omega'$ for some q'_a if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$.

These differ in the placement of the existential quantifier. The first pair are, in fact, false.

For the former: Assume that $q \xrightarrow{a} q''_a$ and $\hat{q}''_a = \Omega' = \hat{q}'_a$. It might be that the states q''_a and q'_a are only bisimilar, not equal. In that case, $q \xrightarrow{a}\!\sim q'_a$ but, in general, not $q \xrightarrow{a} q'_a$ directly.

For the latter: Assume that $q \xrightarrow{a} q''_a$ and $\hat{q}''_a = \Omega'$. Choosing $q'_a := q''_a$, we indeed have $q \xrightarrow{a} q'_a$ and $\hat{q}'_a = \Omega'$.

$$\begin{array}{ccc} q & \xrightarrow{a} & q'_a \\ {\scriptstyle\mathcal{R}}\big| & & \big|{\scriptstyle\mathcal{R}} \\ \underset{\rightarrow}{a}\,\hat{q} & \longrightarrow & \Omega' = \hat{q}'_a \end{array} \qquad \begin{array}{ccc} q & \xrightarrow{a} & q'_a \\ {\scriptstyle\mathcal{R}}\big| & & \vdots{\scriptstyle\mathcal{R}} \\ \underset{\rightarrow}{a}\,\hat{q} & \longrightarrow & \Omega' = \hat{q}'_a \end{array}$$

5.7 *Encoding deterministic finite automata*

Recall from **??** our string rewriting specification of how an NFA processes its input. Given a DFA $\mathcal{A} = (Q, ?, F)$ over an input alphabet Σ, the NFA's operational semantics are adequately captured by the folllwing string rewriting axioms:

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

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

5.7.1 *A functional choreography*

One possible choreography for this specification treats the input symbols $a \in \Sigma$ as atomic propositions $\underset{\rightarrow}{a}$; states $q \in Q$ as recursively defined propostions $\hat{q}$;and the end-of-word marker ϵ as an atomic proposition $\underset{\rightarrow}{\epsilon}$. In other words, the NFA's input is treated as a sequence of messages, $\underset{\rightarrow}{\epsilon}\,\underset{\rightarrow}{a_n} \cdots \underset{\rightarrow}{a_2}\,\underset{\rightarrow}{a_1}$, and the NFA's states are treated as [recursive] processes.

$a \mapsto \underset{\rightarrow}{a}$ for all $a \in \Sigma$; $q \mapsto \hat{q}$ for all $q \in Q$; and $\epsilon \mapsto \underset{\rightarrow}{\epsilon}$.

Using this assignment, the choreography constructed from the specification consists of the following definition, one for each NFA state $q \in Q$:

$$\hat{q} \triangleq \mathop{\&}_{a \in \Sigma} \mathop{\&}_{q'_a} (\underset{\rightarrow}{a} \setminus \hat{q}'_a) \,\&\, (\underset{\rightarrow}{\epsilon} \setminus \hat{F}(q))\,.$$

Corollary 5.14. *If $a\,q \longrightarrow q'_a$, then $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \hat{q}'_a$. If $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$, then $a\,q \longrightarrow w'$ and $\Omega' \Longrightarrow \theta(w')$.*

Corollary 5.15. *If $q \xrightarrow{a} q'_a$, then $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \hat{q}'_a$. If $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a$, then $q \xrightarrow{a} q''_a$ for some q''_a such that $\hat{q}'_a = \hat{q}''_a$.*

As an extended example, we will use ordered rewriting to specify how a DFA processes its input. Given a DFA $\mathcal{A} = (Q, ?, F)$ over an input alphabet Σ, the idea is to encode each state, $q \in Q$, as an ordered proposition, $\hat{q}$, in such a way that the DFA's operational semantics are adequately captured by [ordered] rewriting. [25]

Ideally, DFA transitions $q \xrightarrow{a} q'_a$ would be in bijective correspondence with rewriting steps $a\,\hat{q} \longrightarrow \hat{q}'_a$, where each input symbol a is encoded by a matching [propositional] atom. We will return to the possibility of this kind of tight correspondence in **??**, but, for now, we will content ourselves with a correspondence with traces rather than individual steps, adopting the following desiderata:

- $q \xrightarrow{a} q'_a$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'_a$, for all input symbols $a \in \Sigma$.
- $q \in F$ if, and only if, $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$, where the atom ϵ functions as an end-of-word marker.

Given the reversal (anti-)homomorphism from finite words to ordered contexts defined in the adjacent figure, the first desideratum is subsumed by a third:

- $q \xrightarrow{w} q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, for all finite words $w \in \Sigma^*$.

From these desiderata [and the observation that DFAs' graphs frequently[26] contain cycles], we arrive at the following encoding, in which each state is encoded by one of a collection of mutually recursive definitions:[27]

$$\hat{q} \triangleq \left(\&_{a\in\Sigma}(a \setminus \hat{q}'_a)\right) \,\&\, \left(\epsilon \setminus \hat{F}(q)\right)$$

where

$$q \xrightarrow{a} q'_a, \text{ for all input symbols } a \in \Sigma, \quad \text{and} \quad \hat{F}(q) = \begin{cases} \mathbf{1} & \text{if } q \in F \\ \top & \text{if } q \notin F\,. \end{cases}$$

Just as each state q has exactly one successor for each input symbol a, its encoding, $\hat{q}$, has exactly one input clause, $(a \setminus \cdots)$, for each symbol a.

For a concrete instance of this encoding, recall from chapter 2 the DFA (repeated in the adjacent figure) that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b; that DFA is encoded by the following definitions:

$$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})$$

Indeed, just as the DFA has a transition $q_0 \xrightarrow{b} q_1$, its encoding admits a trace

$$b\,\hat{q}_0 = b\left((a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \top)\right) \Longrightarrow b\,(b \setminus \hat{q}_1) \longrightarrow \hat{q}_1\,.$$

And, just as q_1 is an accepting state, its encoding also admits a trace

$$\epsilon\,\hat{q}_1 = \epsilon\left((a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})\right) \Longrightarrow \epsilon\,(\epsilon \setminus \mathbf{1}) \longrightarrow \mathbf{1}\,.$$

[25] [In general, the behavior of a DFA state is recursive, so the proposition $\hat{q}$ will be recursively defined.]

$$(w_1\,w_2)^{\mathrm{R}} = w_2^{\mathrm{R}}\,w_1^{\mathrm{R}}$$
$$\epsilon^{\mathrm{R}} = \cdot$$
$$a^{\mathrm{R}} = a$$

Figure 5.13: An (anti-)homomorphism for reversal of finite words to ordered contexts

[26] Actually, there is always at least one cycle in a well-formed DFA.

[27] q'_a, using function or relation?

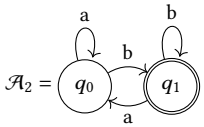


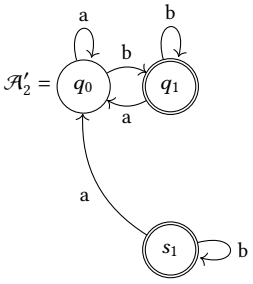
Figure 5.14: A DFA that accepts, from state q_0, exactly those words that end with b. (Repeated from fig. 2.2.)

MORE GENERALLY, this encoding is complete, in the sense that it simulates all DFA transitions: $q \xrightarrow{a} q'$ implies $a\,\hat{q} \Longrightarrow \hat{q}'$, for all states q and q' and input symbols a.

However, the converse does not hold – the encoding is unsound because there are rewritings that cannot be simulated by a DFA transition.

FALSE CLAIM 5.16. *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be a DFA over the input alphabet Σ. Then $a\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{a} q'$, for all input symbols $a \in \Sigma$.*

Counterexample. Consider the DFA and encoding shown in the adjacent figure; it is the same DFA as shown in fig. 5.27, but with one added state, s_1, that is unreachable from q_0 and q_1. Notice that, as a coinductive consequence of the equirecursive treatment of definitions, $\hat{q}_1 = \hat{s}_1$. Previously, we saw that $b\,\hat{q}_0 \Longrightarrow \hat{q}_1$; hence $b\,\hat{q}_0 \Longrightarrow \hat{s}_1$. However, the DFA has no $q_0 \xrightarrow{b} s_1$ transition (because $q_1 \neq s_1$), and so this encoding is unsound with respect to the operational semantics of DFAs. □



$\hat{q}_0 \triangleq (a \setminus \hat{q}_0)\ \&\ (b \setminus \hat{q}_1)\ \&\ (\epsilon \setminus \top)$

$\hat{q}_1 \triangleq (a \setminus \hat{q}_0)\ \&\ (b \setminus \hat{q}_1)\ \&\ (\epsilon \setminus \mathbf{1})$

$\hat{s}_1 \triangleq (a \setminus \hat{q}_0)\ \&\ (b \setminus \hat{s}_1)\ \&\ (\epsilon \setminus \mathbf{1})$

Figure 5.15: fig:ordered-rewriting:dfa-counterexample:dfa A slightly modified version of the DFA from fig. 5.27; and fig:ordered-rewriting:dfa-counterexample:encoding its encoding

As this counterexample shows, the lack of adequacy stems from attempting to use an encoding that is not injective – here, $q_1 \neq s_1$ even though $\hat{q}_1 = \hat{s}_1$. In other words, eqality of state encodings is a coarser eqvivalence than equality of the states themselves.

One possible remedy for this lack of adequacy 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 $\{q \mid q \in F\}$ and $\{\bar{q} \mid q \in Q - F\}$ to tag final and non-final states, respectively, we could define an alternative encoding, $\check{q}$:

$$\check{q} \triangleq \left(\&_{a \in \Sigma}(a \setminus \check{q}'_a)\right)\ \&\ \left(\epsilon \setminus \check{F}(q)\right)$$

where

$$q \xrightarrow{a} q'_a, \text{ for all input symbols } a \in \Sigma, \quad \text{and} \quad \check{F}(q) = \begin{cases} q & \text{if } q \in F \\ \bar{q} & \text{if } q \notin F\,. \end{cases}$$

Under this alternative encoding, the states q_1 and s_1 of fig. 5.28 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 (**??**). In other words, although the encoding is not, strictly speaking, injective, it is injective *up to bisimilarity*: $\hat{q} = \hat{s}$ implies $q \sim s$. This suggests a more elegant solution to the apparent lack of adequacy: the encoding's adequacy should be judged up to DFA bisimilarity.

THEOREM 5.31 (DFA adequacy up to bisimilarity). *Let $\mathcal{A} = (Q, ?, F)$ be a DFA over the input alphabet Σ. Then, for all states q, q', and s:*

1. $q \sim s$ *if, and only if,* $\hat{q} = \hat{s}$.
2. $q \sim\!\xrightarrow{a}\!\sim q'$ *if, and only if,* $a\,\hat{q} \Longrightarrow \hat{q}'$, *for all input symbols* $a \in \Sigma$. *More generally,* $q \sim\!\xrightarrow{w}\!\sim q'$ *if, and only if,* $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, *for all finite words* $w \in \Sigma^*$.
3. $q \in F$ *if, and only if,* $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.

Before proving this theorem, we must first prove a lemma: the only traces from one state's encoding to another's are the trivial traces.

LEMMA 5.17. *Let* $\mathcal{A} = (Q, ?, F)$ *be a DFA over the input alphabet* Σ. *For all states* q *and* s, *if* $\hat{q} \Longrightarrow \hat{s}$, *then* $\hat{q} = \hat{s}$.

Proof. Assume that a trace $\hat{q} \Longrightarrow \hat{s}$ exists. If the trace is trivial, then $\hat{q} = \hat{s}$ is immediate. Otherwise, the trace is nontrivial and consists of a strictly positive number of rewriting steps. By inversion, those rewriting steps drop one or more conjuncts from $\hat{q}$ to form $\hat{s}$. Every DFA state's encoding contains exactly $|\Sigma| + 1$ conjuncts – one for each input symbol a and one for the end-of-word marker, ϵ. If even one conjunct is dropped from $\hat{q}$, not enough conjuncts will remain to form $\hat{s}$. Thus, a nontrivial trace $\hat{q} \Longrightarrow \hat{s}$ cannot exist. □

It is important to differentiate this lemma from the false claim that a state's encoding can take no rewriting steps. There certainly exist nontrivial traces from $\hat{q}$, but they do not arrive at the encoding of any state.

With this lemma now in hand, we can proceed to proving adequacy up to bisimilarity.

THEOREM 5.18 (DFA adequacy up to bisimilarity). *Let* $\mathcal{A} = (Q, ?, F)$ *be a DFA over the input alphabet* Σ. *Then, for all states* q, q', *and* s:

1. $q \sim s$ *if, and only if,* $\hat{q} = \hat{s}$.
2. $q \sim\!\xrightarrow{a}\!\sim q'$ *if, and only if,* $a\,\hat{q} \Longrightarrow \hat{q}'$, *for all input symbols* $a \in \Sigma$. *More generally,* $q \sim\!\xrightarrow{w}\!\sim q'$ *if, and only if,* $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, *for all finite words* $w \in \Sigma^*$.
3. $q \in F$ *if, and only if,* $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.

Proof. Each part is proved in turn. The proof of part 2 depends on the proof of part 1.

1. We shall show that bisimilarity coincides with equality of encodings, proving each direction separately.
   - To prove that bisimilar DFA states have equal encodings – *i.e.*, that $q \sim s$ implies $\hat{q} = \hat{s}$ – a fairly straightforward proof by coinduction suffices.

     Let q and s be bisimilar states. By the definition of bisimilarity (??), two properties hold:
     - For all input symbols a, the unique a-successors of q and s are also bisimilar.
     - States q and s have matching finalities – *i.e.*, $q \in F$ if and only if $s \in F$.

Applying the coinductive hypothesis to the former property, we may deduce that, for all symbols a, the a-successors of q and s also have equal encodings. From the latter property, it follows that $\hat{F}(q) = \hat{F}(s)$. Because definitions are interpreted equirecursively, these equalities together imply that q and s themselves have equal encodings.

- To prove the converse – that states with equal encodings are bisimilar – we will show that the relation $\mathcal{R} = \{(q, s) \mid \hat{q} = \hat{s}\}$, which relates states if they have equal encodings, is a bisimulation and is therefore included in bisimilarity.
  - The relation $\mathcal{R}$ is symmetric.
  - We must show that $\mathcal{R}$-related states have $\mathcal{R}$-related a-successors, for all input symbols a.
    Let q and s be $\mathcal{R}$-related states. Being $\mathcal{R}$-related, q and s have equal encodings; because definitions are interpreted equirecursively, the unrollings of those encodings are also equal. By definition of the encoding, it follows that, for each input symbol a, the unique a-successors of q and s have equal encodings. Therefore, for each a, the a-successors of q and s are themselves $\mathcal{R}$-related.
  - We must show that $\mathcal{R}$-related states have matching finalities.
    Let q and s be $\mathcal{R}$-related states, with q a final state. Being $\mathcal{R}$-related, q and s have equal encodings; because definitions are interpreted equirecursively, the unrollings of those encodings are also equal. It follows that $\hat{F}(q) = \hat{F}(s)$, and so s is also a final state.

2. We would like to prove that $q \sim\!\!\xrightarrow{a}\!\!\sim q'$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'$, or, more generally, that $q \sim\!\!\xrightarrow{w}\!\!\sim q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$. Because bisimilar states have equal encodings (part 1) and bisimilarity is reflexive (**??**), it suffices to show two stronger statements: (a) that $q \xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$; and (b) that $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{w}\!\!\sim q'$. We prove these in turn.

   (a) We shall prove that $q \xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ by induction over the structure of word w.

   - Consider the case of the empty word, ϵ; we must show that $q = q'$ implies $\hat{q} \Longrightarrow \hat{q}'$. Because the encoding is a function, this is immediate.
   - Consider the case of a nonempty word, $a\,w$; we must show that $q \xrightarrow{a}\!\xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow \hat{q}'$. Let q'_a be an a-successor of state q that is itself w-succeeded by state q'. There exists, by definition of the encoding, a trace

     $$w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow w^{\mathrm{R}}\,a\,(a \setminus \hat{q}'_a) \longrightarrow w^{\mathrm{R}}\,\hat{q}'_a \Longrightarrow \hat{q}'\,,$$

     with the trace's tail justified by an appeal to the inductive hypothesis.

   (b) We shall prove that $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{w}\!\!\sim q'$ by induction over the structure of word w.

- Consider the case of the empty word, ϵ; we must show that $\hat{q} \Longrightarrow \hat{q}'$ implies $q \sim q'$. By lemma 5.30, $\hat{q} \Longrightarrow \hat{q}'$ implies that q and q' have equal encodings. Part 1 can then be used to establish that q and q' are bisimilar.
- Consider the case of a nonempty word, $a\,w$; we must show that $w^{\mathrm{R}}\, a\, \hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{a} \xrightarrow{w} \sim q'$. By inversion[28], the given trace can only begin by inputting a:

  $$w^{\mathrm{R}}\, a\, \hat{q} \Longrightarrow w^{\mathrm{R}}\, a\, (a \setminus \hat{q}'_a) \longrightarrow w^{\mathrm{R}}\, \hat{q}'_a \Longrightarrow \hat{q}'\,,$$

  where q'_a is an a-successor of state q. An appeal to the inductive hypothesis on the trace's tail yields $q'_a \xrightarrow{w} \sim q'$, and so the DFA admits $q \xrightarrow{a} \xrightarrow{w} \sim q'$, as required.

[28] Is this enough justification?

3. We shall prove that the final states are exactly those states q such that $\epsilon\, \hat{q} \Longrightarrow \mathbf{1}$.
   - Let q be a final state; accordingly, $\hat{F}(q) = \mathbf{1}$. There exists, by definition of the encoding, a trace

     $$\epsilon\, \hat{q} \Longrightarrow \epsilon\, (\epsilon \setminus \hat{F}(q)) \longrightarrow \hat{F}(q) = \mathbf{1}\,.$$

   - Assume that a trace $\epsilon\, \hat{q} \Longrightarrow \mathbf{1}$ exists. By inversion[29], this trace can only begin by inputting ϵ:

     $$\epsilon\, \hat{q} \Longrightarrow \epsilon\, (\epsilon \setminus \hat{F}(q)) \longrightarrow \hat{F}(q) \Longrightarrow \mathbf{1}\,.$$

     The tail of this trace, $\hat{F}(q) \Longrightarrow \mathbf{1}$, can exist only if q is a final state. □

[29] Is this enough justification?

5.7.2 *Encoding nondeterministic finite automata?*

We would certainly be remiss if we did not attempt to generalize the rewriting specification of DFAs to one for their nondeterministic cousins.

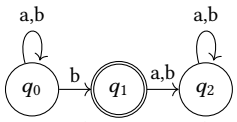
Differently from DFA states, an NFA state q may have several nondeterministic successors for each input symbol a. To encode the NFA state q, all of its a-successors are collected in an alternative conjunction underneath the left-handed input of a. Thus, the encoding of an NFA state q becomes

$$\hat{q} \triangleq \left(\mathop{\&}_{a \in \Sigma} \left(a \setminus \left(\&_{q'_a}\, \hat{q}'_a\right)\right)\right) \,\&\, \left(\epsilon \setminus \hat{F}(q)\right),$$

where $\hat{F}(q)$ is defined as for DFAs.

The adjacent figure recalls from chapter 2 an NFA that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b. Using the above encoding of NFAs, ordered rewriting does indeed simulate this NFA. For example, just as there are transitions $q_0 \xrightarrow{b} q_0$ and $q_0 \xrightarrow{b} q_1$, there are traces

$$b\, \hat{q}_0 \Longrightarrow b\, \left(b \setminus (\hat{q}_0 \,\&\, \hat{q}_1)\right) \longrightarrow \hat{q}_0 \,\&\, \hat{q}_1 \begin{array}{l} \nearrow\, \hat{q}_0 \\ \searrow\, \hat{q}_1 \end{array}$$



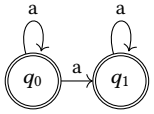
$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \,\&\, \left(b \setminus (\hat{q}_0 \,\&\, \hat{q}_1)\right) \,\&\, (\epsilon \setminus \top)$
$\hat{q}_1 \triangleq (a \setminus \hat{q}_2) \,\&\, (b \setminus \hat{q}_2) \,\&\, (\epsilon \setminus \mathbf{1})$
$\hat{q}_2 \triangleq (a \setminus \hat{q}_2) \,\&\, (b \setminus \hat{q}_2) \,\&\, (\epsilon \setminus \top)$

Figure 5.16: fig:ordered-rewriting:nfa-example:nfa An NFA that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b; and fig:ordered-rewriting:nfa-example:encoding its encoding

Unfortunately, while it does simulate NFA behavior, this encoding is not adequate. Like DFA states, NFA states that have equal encodings are bisimilar. However, for NFAs, the converse does not hold: bisimilar states do not necessarily have equal encodings.

FALSE CLAIM 5.19. *Let $\mathcal{A} = (Q, ?, F)$ be an NFA over input alphabet Σ. Then $q \sim s$ implies $\hat{q} = \hat{s}$, for all states q and s.*

Counterexample. Consider the NFA and encoding depicted in the adjacent figure. It is easy to verify that the relation $\{q_1\} \times \{q_0, q_1\}$ is a bisimulation; in particular, q_1 simulates the $q_0 \xrightarrow{a} q_1$ transition by its self-loop, $q_1 \xrightarrow{a} q_1$. Hence, q_0 and s_0 are bisimilar. It is equally easy to verify, by unrolling the definitions used in the encoding, that $\hat{q}_0 \neq \hat{s}_0$. □



$$\hat{q}_0 \triangleq (a \setminus (\hat{q}_0 \,\&\, \hat{q}_1)) \,\&\, (\epsilon \setminus \mathbf{1})$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})$$

Figure 5.17: An NFA that accepts all finite words over the alphabet $\Sigma = \{a\}$

For DFAs, bisimilar states do have equal encodings because the inherent determinism DFA bisimilarity is a rather fine-grained equivalence. Because each DFA state has exactly one successor for each input symbol The additional flexibility entailed by nondeterminism

Once again, it would be possible to construct an adequate encoding, by tagging each state with a unique atom.

For the moment, we will put aside the question of an adequate encoding of NFAs.

5.8 *Introduction*

In the previous chapter, we saw that the ordered sequent calculus can be given a resource interpretation in which sequents $\Omega \vdash A$ may be read as "From resources Ω, resource goal A is achievable." For instance, the left rule for ordered conjunction ($\bullet$L, see adjacent display) was read "Goal C is achievable from resource $A \bullet B$ if it is achievable from the separate resources $A\,B$."

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

As alluded in the previous chapter's discussion of ordered conjunction[30], this $\bullet$L rule is essentially a rule of resource decomposition: it decomposes [the resource] $A \bullet B$ into the separate resources $A\,B$ and relegates the unchanged goal C to a secondary role.

[30] See page 32.

THIS CHAPTER begins by exploring a refactoring of the ordered sequent calculus's left rules around this idea of resource decomposition (??). Most of the left rules can be easily refactored in this way, although a few will prove resistant to the change.

Emphasizing resource decomposition naturally leads us to a rewriting interpretation of (a fragment of) ordered logic (??). This rewriting system is closely related to traditional notions of string rewriting,[31] but simultaneously restricts and generalizes [...] along distinct axes.

[31] ??.

The connection of ordered logic and the Lambek calculus to rewriting is certainly not new. Lambek's original article[32]

[32] Lambek 1958.

This development borrows from Cervesato and Scedrov's work on intuitionistic linear logic as an expressive rewriting framework that generalizes traditional notions of multiset rewriting.[33]

[33] Cervesato and Scedrov 2009.

Most of the left rules could be seen as decomposing resources. The left rules were seen as decomposing resources, such as the $\bullet$L rule decomposing $A \bullet B$ into the resources $A\,B$. The right rules, on the other hand, were seen as ...

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

Replacing the left rules with a single, common rule ... and a new judgment, $\Omega \longrightarrow \Omega'$, that exposes [makes [more] explicit] the decomposition of resources/state transformation aspect.

5.9 *Most left rules decompose ordered resources*

Recall two of the ordered sequent calculus's left rules: $\bullet$L and $\&\text{L}_1$.

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

Both rules decompose the principal resource: in the $\bullet$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 string of resources], without restriction; these concerns are not specific to the $\bullet$L and $\&\text{L}_1$ rules, but are general boilerplate that arguably should be factored out.

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

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

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

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

$$\leftrightsquigarrow$$

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

Figure 5.18: A refactoring of the $\bullet$L rule as 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 close, we could introduce the decomposition rules

$$\frac{\Omega \vdash A}{\Omega\, (A \setminus B) \longrightarrow B}\ \setminus\text{D}' \qquad \text{and} \qquad \frac{\Omega \vdash A}{(B \mathbin{/} A)\, \Omega \longrightarrow B}\ /\text{D}'.$$

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

$$\rightsquigarrow$$

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

Figure 5.19: A refactoring of the $\setminus$L rule using a resource decomposition rule

Just as for ordered conjunction, the left rules for left- and right-handed implication are then recovered by combining a decomposition rule with the CUT$^{\longrightarrow}$ rule (see adjacent figure).

Although these $\setminus$D′ and /D′ 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 $\setminus$D′ and /D′ rules have premises, and those premises create a dependence of the decomposition judgment upon general provability. Second, the above $\setminus$D′ and /D′ rules do not decompose the principal proposition into immediate subformulas. This contrasts with, for example, the $\bullet$D rule that decomposes $A \bullet B$ into the immediate subformulas $A\, B$.

For these reasons, the above $\setminus$D′ and /D′ rules are somewhat undesirable. Fortunately, there is an alternative. Filling in the $\Omega \vdash A$ premises with the ID^A rule, we arrive at the derivable rules

$$\frac{}{A\, (A \setminus B) \longrightarrow B}\ \setminus\text{D} \qquad \text{and} \qquad \frac{}{(B \,/\, A)\, A \longrightarrow B}\ /\text{D}.$$

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

$$\rightsquigarrow$$

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

Figure 5.20: A refactoring of the $\setminus$L rule using an alternative resource decomposition rule

The standard $\setminus$L and /L rules can still be recovered from these more specific decomposition rules, thanks to 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.[34]

[34] Lambek 1958.

So, 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, $\oplus$L, doesn't decompose the resource $A \oplus B$. The $\oplus$L rule certainly does decompose $A \oplus B$, but it does so [...]. $A \oplus B \longrightarrow A \mid B$ [...] retain the standard $\oplus$L and $\mathbf{0}$L rules.

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

FIGURE 5.21 PRESENTS the fully refactored sequent calculus for ordered logic. This refactored calculus is sound and complete with respect to the ordered sequent calculus (fig. 3.2).

THEOREM 5.20 (Soundness). *If $\Omega \vdash A$ is derivable in the refactored calculus of fig. 5.21, then $\Omega \vdash A$ is derivable in the ordered sequent calculus (fig. 3.2).*

Proof. By structural induction on the given derivation. The key lemma is the admissibility of CUT$^{\longrightarrow}$ in the 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. 5.18 and 5.20. □

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

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

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

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

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

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

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

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

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

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

Figure 5.21: A refactoring of the ordered sequent calculus to emphasize that most left rules amount to resource decomposition

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

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

$$\frac{}{A\,(A \setminus B) \longrightarrow B}\ \setminus\text{D} \qquad \frac{}{(B \,/\, A)\,A \longrightarrow B}\ /\text{D}$$

(no $\oplus$D and $\mathbf{0}$D rules)

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1'\,\Omega_2}\ \longrightarrow\text{C}_\text{L} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1\,\Omega_2'}\ \longrightarrow\text{C}_\text{R}$$

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

Figure 5.22: A rewriting fragment of ordered logic, based on resource decomposition

THEOREM 5.21 (Completeness). *If $\Omega \vdash A$ is derivable in the ordered sequent calculus (fig. 3.2), then $\Omega \vdash A$ is derivable in the refactored calculus of fig. 5.21.*

Proof. 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. 5.18 and 5.20. □

5.10 *Decomposition as rewriting*

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. Instead, we can also 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'."

Figure 5.22 summarizes the rewriting system that we obtain from the refactored sequent-like calculus of fig. 5.21. Essentially, the ordered rewriting system 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 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. 5.21, this kind of framing is taken care of by the cut principle for decomposition, CUT$^{\longrightarrow}$. To express framing at the level of the $\Omega \longrightarrow \Omega'$ judgment, we introduce two compatibility rules: together,

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1'\,\Omega_2}\ \longrightarrow\text{C}_\text{L} \qquad \text{and} \qquad \frac{\Omega_2 \longrightarrow \Omega_2'}{\Omega_1\,\Omega_2 \longrightarrow \Omega_1\,\Omega_2'}\ \longrightarrow\text{C}_\text{R}$$

ensure that rewriting is compatible with concatenation of ordered contexts.[35]

[35] Because ordered contexts form a monoid, these compatibility rules are equivalent to the unified rule

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

However, we prefer the two-rule formulation of compatibility because it better aligns with the syntactic structure of contexts.

By forming the reflexive, transitive closure of $\longrightarrow$, we may construct a multi-step rewriting relation, which we choose to write as $\Longrightarrow$.[36]

Consistent with its [free] monoidal structure, there are two equivalent formulations of this reflexive, transitive closure: each rewriting sequence $\Omega \Longrightarrow \Omega'$ can be viewed as either a list or tree of individual rewriting steps. We prefer the list-based formulation shown in fig. 5.22 because it tends to [...] proofs by structural induction, but, on the basis of the following **??**, we allow ourselves to freely switch between the two formulations as needed.

FACT 5.22 (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 $\backslash\text{D}$ and $/\text{D}$ rules. The $\backslash\text{D}'$ and $/\text{D}'$ rules, with their $\Omega \vdash A$ premises, would be problematic as rewriting rules because they would introduce a dependence of ordered rewriting upon general provability, and the concomitant[/attendant] proof search would take ordered rewriting too far afield from traditional, syntactic[37] notions of string and multiset rewriting. [mechanical, computational]

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

FALSE CLAIM 5.23 (Completeness). *If* $\Omega \vdash A$, *then* $\Omega \Longrightarrow A$.

Counterexample. The sequent $A \setminus (C \,/\, B) \vdash (A \setminus C) \,/\, B$ is provable, but $A \setminus (C \,/\, B) \Longrightarrow (A \setminus C) \,/\, B$ 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. Before stating and proving soundness, we must define an operation $\bullet\Omega$ that reifies an ordered context as a single proposition (see adjacent figure).

THEOREM 5.25 (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. □

Last, notice that every rewriting step, $\Omega \longrightarrow \Omega'$, strictly decreases the number of logical connectives that occur in the ordered context. More formally, let $|\Omega|$ 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 **??**.

FACT 5.26. *If* $\Omega \longrightarrow \Omega'$, *then* $|\Omega| > |\Omega'|$. *Also, if* $\Omega \Longrightarrow \Omega'$, *then* $|\Omega| \geq |\Omega'|$.

Proof. By induction on the structure of the rewriting step. □

On the basis of this **??**, we will frequently refer to the rewriting relation, $\longrightarrow$, as reduction.

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

[37] Is this the right word?

$$(\Omega_1\ \Omega_2) = (\Omega_1) \bullet (\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$A = A$$

$$\bullet(\Omega_1\ \Omega_2) = (\bullet\Omega_1) \bullet (\bullet\Omega_2)$$
$$\bullet(\cdot) = \mathbf{1}$$
$$\bullet A = A$$

THEOREM 5.24. *If* $\Omega \longrightarrow \Omega'$, *then* $\Omega \vdash \bullet\Omega'$. *Also, if* $\Omega \Longrightarrow \Omega'$, *then* $\Omega \vdash \bullet\Omega'$.

Figure 5.23: From ordered contexts to propositions

$$|\Omega_1\ \Omega_2| = |\Omega_1| + |\Omega_2|$$
$$|\cdot| = 0$$
$$|A \star B| = 1 + |A| + |B| \quad \text{if } \star = \bullet, \&, \backslash, /, \text{ or } \oplus$$
$$|A| = 1 \quad \text{if } A = \alpha, \mathbf{1}, \top, \text{ or } \mathbf{0}$$

Figure 5.24: A measure of the number of logical connectives within an ordered context

5.11

5.11.1 *Binary counters*

$$\hat{e}\,\underset{\leftarrow}{i} \Longrightarrow \hat{e}\,\hat{b}_1$$
$$\hat{b}_0\,\underset{\leftarrow}{i} \Longrightarrow \hat{b}_1$$
$$\hat{b}_1\,\underset{\leftarrow}{i} \Longrightarrow \underset{\leftarrow}{i}\,\hat{b}_0$$

$$\hat{e}\,\underset{\leftarrow}{d} \Longrightarrow \underset{\rightarrow}{z}$$
$$\hat{b}_0\,\underset{\leftarrow}{d} \Longrightarrow \underset{\leftarrow}{d}\,\hat{b}'_0$$
$$\hat{b}_1\,\underset{\leftarrow}{d} \Longrightarrow \hat{b}_0\,\underset{\rightarrow}{s}$$
$$\underset{\rightarrow}{z}\,\hat{b}'_0 \Longrightarrow \underset{\rightarrow}{z}$$
$$\underset{\rightarrow}{s}\,\hat{b}'_0 \Longrightarrow \hat{b}_1\,\underset{\rightarrow}{s}$$

$$\frac{}{e\ \mathcal{R}\ \hat{e}} \qquad \frac{\Omega\ \mathcal{R}\ \Omega'}{\Omega\,b_0\ \mathcal{R}\ \Omega'\,\hat{b}_0} \qquad \frac{\Omega\ \mathcal{R}\ \Omega'}{\Omega\,b_1\ \mathcal{R}\ \Omega'\,\hat{b}_1} \qquad \frac{\Omega\ \mathcal{R}\ \Omega'}{\Omega\,i\ \mathcal{R}\ \Omega'\,\underset{\leftarrow}{i}}$$

$$\frac{\Omega\ \mathcal{R}\ \Omega'}{\Omega\,i\,b_0\ \mathcal{R}\ \Omega'\,(\underset{\leftarrow}{i} \bullet b_0)}$$

5.11.2 *Automata*

$$\frac{\Omega\ \mathcal{R}\ \Omega'}{a\,\Omega\ \mathcal{R}\ \underset{\rightarrow}{a}\,\Omega'} \qquad \frac{}{q\ \mathcal{R}\ \hat{q}}$$

$$\frac{q \sim q'}{q\ \mathcal{R}\ \hat{q}'}$$

5.12 *Ordered rewriting for specifications*

5.12.1 *Deterministic finite automata*

$$\frac{}{a\,q \longrightarrow q'_a}$$

for each DFA transition $q \xrightarrow{a} q'_a$, and

$$\frac{}{\epsilon\,q \longrightarrow F(q)}$$

for each state q, where $F(q) = \mathbf{1}$ if q is a final state and $F(q) = \top$ otherwise.

- $q \xrightarrow{a} q'_a$ if, and only if, $a\,q \longrightarrow q'_a$; and
- $q \in F$ if, and only if, $\epsilon\,q \longrightarrow \mathbf{1}$.

5.12.2 *Nondeterministic finite automata*

Equally straightforward

5.12.3 *Binary counters*

Values

AN INCREMENT OPERATION To use ordered rewriting to specify [...]

$$\overline{e\ i \longrightarrow e\ b_1} \qquad \overline{b_0\ i \longrightarrow b_1} \qquad \overline{b_1\ i \longrightarrow i\ b_0}$$

Small- and big-step adequacy theorems for increments

- Slightly simplified because there is no •

A DECREMENT OPERATION

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

- Significantly simpler because there is no &, so we don't need (weak) focusing

5.13

5.13.1 *Concurrency in ordered rewriting*

As an example of multi-step rewriting, observe that

$$\alpha_1\ (\alpha_1 \setminus \alpha_2)\ (\beta_2 \,/\, \beta_1)\ \beta_1 \Longrightarrow \alpha_2\ \beta_2.$$

In fact, as shown in the adjacent figure, two sequences witness this rewriting: either the initial state's left half, $\alpha_1\ (\alpha_1 \setminus \alpha_2)$, is first rewritten to α_2 and then its right half, $(\beta_2 \,/\, \beta_1)\ \beta_1$, is rewritten to β_2; or *vice versa*, the right half is first rewritten to β_2 and then the left half is rewritten to α_2.

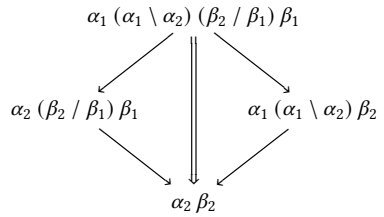


Figure 5.25: An example of concurrent ordered rewriting

Notice that these two sequences differ only in how non-overlapping, and therefore independent, rewritings of the initial state's two halves are interleaved. Consequently, 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 $\alpha_1\ (\alpha_1 \setminus \alpha_2)\ (\beta_2 \,/\, \beta_1)\ \beta_1$ to $\alpha_2\ \beta_2$ remains.

More generally, this idea that the interleaving of independent actions is irrelevant is known as *concurrent equality*,[38] and it forms the basis of concurrency.[39] Concurrent equality also endows traces $\Omega \Longrightarrow \Omega'$ with a free partially commutative monoid structure, *i.e.*, traces form a trace monoid.

Because the two indivisual rewriting steps are independent, Nothing about the final result, $\alpha_2\ \beta_2$, suggests which rewriting sequence

The rewritings of the left and right halves are not overlapping and therefore independent. Their independence means that we may view the two rewriting sequences as equivalent – the two rewriting steps

More generally, any non-overlapping rewritings are independent and may occur in any order. Rewriting sequences that differ only by the order in which

[38] **Watkins+:CMU02**.

[39] **??**.

independent rewritings occur may be seen as equivalent sequences. This equivalence relation, *concurrent equality*[40]

because the left half of Ω may be rewritten by the $\backslash\text{D}$ rule to α_2, and then the right half may be rewritten to β_2:

[40] **Watkins+:CMU02.**

5.13.2 *Other properties of ordered rewriting*

As the relation $\Longrightarrow$ forms a rewriting system, we may evaluate it along several standard dimensions: termination, confluence.

Because each rewriting step reduces the number of logical connectives present in the state (**??** 5.26), ordered rewriting is terminating.

THEOREM 5.27 (Termination). *No infinite rewriting sequence* $\Omega_0 \longrightarrow \Omega_1 \longrightarrow \Omega_2 \longrightarrow \cdots$ *exists.*

Proof. Beginning from state Ω_0, some state Ω_i will eventually be reached such that either: $\Omega_i \not\longrightarrow$; or $|\Omega_i| = 0$ and $\Omega_i \longrightarrow \Omega_{i+1}$. In the latter case, **??** 5.26 establishes $|\Omega_{i+1}| < 0$, which is impossible. □

Although terminating, ordered rewriting is not confluent. Confluence requires that all states with a common ancestor, *i.e.*, states Ω_1' and Ω_2' such that $\Omega_1' \Longleftarrow\!\!\Longrightarrow \Omega_2'$, be joinable, *i.e.*, $\Omega_1' \Longrightarrow\!\!\Longleftarrow \Omega_2'$. Because ordered rewriting is directional[41] and the relation $\Longrightarrow$ is not symmetric, some nondeterministic choices are irreversible.

[41] Is this phrasing correct?

FALSE CLAIM 5.28 (Confluence). *If* $\Omega_1' \Longleftarrow\!\!\Longrightarrow \Omega_2'$, *then* $\Omega_1' \Longrightarrow\!\!\Longleftarrow \Omega_2'$.

Counterexamples. Consider the state $\alpha \& \beta$. By the rewriting rules for additive conjunction, $\alpha \longleftarrow \alpha \,\&\, \beta \longrightarrow \beta$, and hence $\alpha \Longleftarrow \alpha \,\&\, \beta \Longrightarrow \beta$. However, being atoms, neither α nor β reduces. And $\alpha \neq \beta$, so $\alpha \Longrightarrow\!\!\Longleftarrow \beta$ does *not* hold.

Even in the $\&$-free fragment, ordered rewriting is not confluent. For example,

$$\not\longleftarrow \beta_1 \,(\alpha \setminus \beta_2) \Longleftarrow (\beta_1 \,/\, \alpha)\, \alpha \,(\alpha \setminus \beta_2) \Longrightarrow (\beta_1 \,/\, \alpha)\, \beta_2 \not\longrightarrow .$$ □

5.14 *Unbounded ordered rewriting*

[42]

[42] Aranda et al. 2007.

Although a seemingly pleasant property, termination (**??**) significantly limits the expressiveness of ordered rewriting. For example, without unbounded rewriting, we cannot even give ordered rewriting specifications of producer-consumer systems or finite automata.

As the proof of termination shows, rewriting is bounded precisely because states consist of finitely many finite propositions. To admit unbounded rewriting, we therefore choose to permit infinite propositions in the form of mutually recursive definitions, $\alpha \triangleq A$. These definitions are collected into a signature, $\Sigma = (\alpha_i \triangleq A_i)_i$, which indexes the rewriting relations: $\longrightarrow_\Sigma$ and $\Longrightarrow_\Sigma$.[43]

[43] We frequently elide the indexing signature, as it is usually clear from context.

To rule out definitions like $\alpha \triangleq \alpha$ that do not correspond to sensible infinite propositions, we also require that definitions be *contractive*[44] – *i.e.*, that the body of each recursive definition begin with a logical connective at the top level.

[44] Gay and Hole 2005.

By analogy with recursive types from functional programming,[45] we must now decide whether to treat definitions *iso*recursively or *equi*recursively. Under an equirecursive interpretation, definitions $\alpha \triangleq A$ may be silently unrolled or rolled at will; in other words, α is literally *equal* to its unrolling, A. In contrast, under an isorecursive interpretation, unrolling a recursively defined proposition would count as an explicit step of rewriting – $\alpha \longrightarrow A$, for example.

[45] ??.

We choose to interpret definitions equirecursively because the equirecursive treatment, with its generous notion of equality, helps to minimize the overhead of recursively defined propositions. As a simple example, under the equirecursive definition $\beta \triangleq a \setminus \beta$, we have the trace

$$a\, a\, \beta = a\, a\, (a \setminus \beta) \longrightarrow a\, \beta = a\, (a \setminus \beta) \longrightarrow \beta$$

or, more concisely, $a\, a\, \beta \longrightarrow a\, \beta \longrightarrow \beta$. Had we chosen an isorecursive treatment of the same definition, we would have only the more laborious

$$a\, a\, \beta \longrightarrow a\, a\, (a \setminus \beta) \longrightarrow a\, \beta \longrightarrow a\, (a \setminus \beta) \longrightarrow \beta.$$

5.14.1 *Replication*

In Milner's development of the π-calculus, there are two avenues to unbounded process behavior: recursive process definitions and replication.

5.15 *Extended examples of ordered rewriting*

5.15.1 *Encoding deterministic finite automata*

As an extended example, we will use ordered rewriting to specify how a DFA processes its input. Given a DFA $\mathcal{A} = (Q, ?, F)$ over an input alphabet Σ, the idea is to encode each state, $q \in Q$, as an ordered proposition, $\hat{q}$, in such a way that the DFA's operational semantics are adequately captured by [ordered] rewriting. [46]

[46] [In general, the behavior of a DFA state is recursive, so the proposition $\hat{q}$ will be recursively defined.]

Ideally, DFA transitions $q \xrightarrow{a} q'_a$ would be in bijective correspondence with rewriting steps $a\, \hat{q} \longrightarrow \hat{q}'_a$, where each input symbol a is encoded by a matching [propositional] atom. We will return to the possibility of this kind of tight correspondence in **??**, but, for now, we will content ourselves with a correspondence with traces rather than individual steps, adopting the following desiderata:

- $q \xrightarrow{a} q'_a$ if, and only if, $a\, \hat{q} \Longrightarrow \hat{q}'_a$, for all input symbols $a \in \Sigma$.
- $q \in F$ if, and only if, $\epsilon\, \hat{q} \Longrightarrow \mathbf{1}$, where the atom ϵ functions as an end-of-word marker.

Given the reversal (anti-)homomorphism from finite words to ordered contexts defined in the adjacent figure, the first desideratum is subsumed by a third:

- $q \xrightarrow{w} q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, for all finite words $w \in \Sigma^*$.

From these desiderata [and the observation that DFAs' graphs frequently[47] contain cycles], we arrive at the following encoding, in which each state is encoded by one of a collection of mutually recursive definitions:[48]

$$\hat{q} \triangleq \left(\&_{a\in\Sigma}(a \setminus \hat{q}'_a)\right) \,\&\, \left(\epsilon \setminus \hat{F}(q)\right)$$

where

$$q \xrightarrow{a} q'_a, \text{ for all input symbols } a \in \Sigma, \quad \text{and} \quad \hat{F}(q) = \begin{cases} \mathbf{1} & \text{if } q \in F \\ \top & \text{if } q \notin F\,. \end{cases}$$

Just as each state q has exactly one successor for each input symbol a, its encoding, $\hat{q}$, has exactly one input clause, $(a \setminus \cdots)$, for each symbol a.

For a concrete instance of this encoding, recall from chapter 2 the DFA (repeated in the adjacent figure) that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b; that DFA is encoded by the following definitions:

$$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})$$

Indeed, just as the DFA has a transition $q_0 \xrightarrow{b} q_1$, its encoding admits a trace

$$b\,\hat{q}_0 = b\,\big((a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \top)\big) \Longrightarrow b\,(b \setminus \hat{q}_1) \longrightarrow \hat{q}_1\,.$$

And, just as q_1 is an accepting state, its encoding also admits a trace

$$\epsilon\,\hat{q}_1 = \epsilon\,\big((a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})\big) \Longrightarrow \epsilon\,(\epsilon \setminus \mathbf{1}) \longrightarrow \mathbf{1}\,.$$

More generally, this encoding is complete, in the sense that it simulates all DFA transitions: $q \xrightarrow{a} q'$ implies $a\,\hat{q} \Longrightarrow \hat{q}'$, for all states q and q' and input symbols a.

However, the converse does not hold – the encoding is unsound because there are rewritings that cannot be simulated by a DFA transition.

False claim 5.29. *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be a DFA over the input alphabet Σ. Then $a\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{a} q'$, for all input symbols $a \in \Sigma$.*

Counterexample. Consider the DFA and encoding shown in the adjacent figure; it is the same DFA as shown in fig. 5.27, but with one added state, s_1, that is unreachable from q_0 and q_1. Notice that, as a coinductive consequence of the equirecursive treatment of definitions, $\hat{q}_1 = \hat{s}_1$. Previously, we saw that $b\,\hat{q}_0 \Longrightarrow \hat{q}_1$; hence $b\,\hat{q}_0 \Longrightarrow \hat{s}_1$. However, the DFA has no $q_0 \xrightarrow{b} s_1$ transition (because $q_1 \neq s_1$), and so this encoding is unsound with respect to the operational semantics of DFAs. □

$$(w_1\,w_2)^{\mathrm{R}} = w_2^{\mathrm{R}}\,w_1^{\mathrm{R}}$$
$$\epsilon^{\mathrm{R}} = \cdot$$
$$a^{\mathrm{R}} = a$$

Figure 5.26: An (anti-)homomorphism for reversal of finite words to ordered contexts

[47] Actually, there is always at least one cycle in a well-formed DFA.

[48] q'_a, using function or relation?

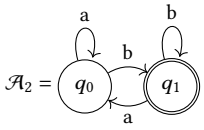
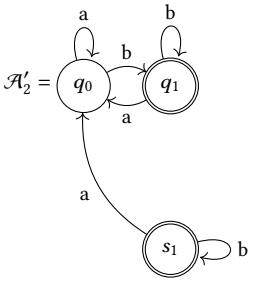


Figure 5.27: A DFA that accepts, from state q_0, exactly those words that end with b. (Repeated from fig. 2.2.)



$$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{q}_1) \,\&\, (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_1 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus \hat{s}_1) \,\&\, (\epsilon \setminus \mathbf{1})$$

Figure 5.28: fig:ordered-rewriting:dfa-counterexample:dfa A slightly modified version of the DFA from fig. 5.27; and fig:ordered-rewriting:dfa-counterexample:encoding its encoding

As this counterexample shows, the lack of adequacy stems from attempting to use an encoding that is not injective – here, $q_1 \neq s_1$ even though $\hat{q}_1 = \hat{s}_1$. In other words, eqality of state encodings is a coarser eqvivalence than equality of the states themselves.

One possible remedy for this lack of adequacy 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 $\{q \mid q \in F\}$ and $\{\bar{q} \mid q \in Q - F\}$ to tag final and non-final states, respectively, we could define an alternative encoding, $\check{q}$:

$$\check{q} \triangleq \left(\&_{a \in \Sigma}(a \setminus \check{q}'_a)\right) \,\&\, \left(\epsilon \setminus \check{F}(q)\right)$$

where

$$q \xrightarrow{a} q'_a, \text{ for all input symbols } a \in \Sigma, \quad \text{and} \quad \check{F}(q) = \begin{cases} q & \text{if } q \in F \\ \bar{q} & \text{if } q \notin F\,. \end{cases}$$

Under this alternative encoding, the states q_1 and s_1 of fig. 5.28 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 (**??**). In other words, although the encoding is not, strictly speaking, injective, it is injective *up to bisimilarity*: $\hat{q} = \hat{s}$ implies $q \sim s$. This suggests a more elegant solution to the apparent lack of adequacy: the encoding's adequacy should be judged up to DFA bisimilarity.

THEOREM 5.31 (DFA adequacy up to bisimilarity). *Let $\mathcal{A} = (Q, ?, F)$ be a DFA over the input alphabet Σ. Then, for all states q, q', and s:*

1. *$q \sim s$ if, and only if, $\hat{q} = \hat{s}$.*
2. *$q \xrightarrow{a}\!\sim\, q'$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'$, for all input symbols $a \in \Sigma$. More generally, $q \xrightarrow{w}\!\sim\, q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, for all finite words $w \in \Sigma^*$.*
3. *$q \in F$ if, and only if, $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.*

Before proving this theorem, we must first prove a lemma: the only traces from one state's encoding to another's are the trivial traces.

LEMMA 5.30. *Let $\mathcal{A} = (Q, ?, F)$ be a DFA over the input alphabet Σ. For all states q and s, if $\hat{q} \Longrightarrow \hat{s}$, then $\hat{q} = \hat{s}$.*

Proof. Assume that a trace $\hat{q} \Longrightarrow \hat{s}$ exists. If the trace is trivial, then $\hat{q} = \hat{s}$ is immediate. Otherwise, the trace is nontrivial and consists of a strictly positive number of rewriting steps. By inversion, those rewriting steps drop one or more conjuncts from $\hat{q}$ to form $\hat{s}$. Every DFA state's encoding contains exactly $|\Sigma| + 1$ conjuncts – one for each input symbol a and one for the end-of-word

marker, ϵ. If even one conjunct is dropped from $\hat{q}$, not enough conjuncts will remain to form $\hat{s}$. Thus, a nontrivial trace $\hat{q} \Longrightarrow \hat{s}$ cannot exist. □

It is important to differentiate this lemma from the false claim that a state's encoding can take no rewriting steps. There certainly exist nontrivial traces from $\hat{q}$, but they do not arrive at the encoding of any state.

With this lemma now in hand, we can proceed to proving adequacy up to bisimilarity.

THEOREM 5.31 (DFA adequacy up to bisimilarity). *Let $\mathcal{A} = (Q, ?, F)$ be a DFA over the input alphabet Σ. Then, for all states q, q', and s:*

1. *$q \sim s$ if, and only if, $\hat{q} = \hat{s}$.*
2. *$q \sim\!\!\xrightarrow{a}\!\!\sim q'$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'$, for all input symbols $a \in \Sigma$. More generally, $q \sim\!\!\xrightarrow{w}\!\!\sim q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, for all finite words $w \in \Sigma^*$.*
3. *$q \in F$ if, and only if, $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.*

Proof. Each part is proved in turn. The proof of part 2 depends on the proof of part 1.

1. We shall show that bisimilarity coincides with equality of encodings, proving each direction separately.
   - To prove that bisimilar DFA states have equal encodings – *i.e.*, that $q \sim s$ implies $\hat{q} = \hat{s}$ – a fairly straightforward proof by coinduction suffices.

     Let q and s be bisimilar states. By the definition of bisimilarity (**??**), two properties hold:
     - For all input symbols a, the unique a-successors of q and s are also bisimilar.
     - States q and s have matching finalities – *i.e.*, $q \in F$ if and only if $s \in F$.

     Applying the coinductive hypothesis to the former property, we may deduce that, for all symbols a, the a-successors of q and s also have equal encodings. From the latter property, it follows that $\hat{F}(q) = \hat{F}(s)$. Because definitions are interpreted equirecursively, these equalities together imply that q and s themselves have equal encodings.
   - To prove the converse – that states with equal encodings are bisimilar – we will show that the relation $\mathcal{R} = \{(q, s) \mid \hat{q} = \hat{s}\}$, which relates states if they have equal encodings, is a bisimulation and is therefore included in bisimilarity.
     - The relation $\mathcal{R}$ is symmetric.
     - We must show that $\mathcal{R}$-related states have $\mathcal{R}$-related a-successors, for all input symbols a.

       Let q and s be $\mathcal{R}$-related states. Being $\mathcal{R}$-related, q and s have equal encodings; because definitions are interpreted equirecursively, the unrollings of those encodings are also equal. By definition of the

encoding, it follows that, for each input symbol a, the unique a-successors of q and s have equal encodings. Therefore, for each a, the a-successors of q and s are themselves $\mathcal{R}$-related.

– We must show that $\mathcal{R}$-related states have matching finalities.
Let q and s be $\mathcal{R}$-related states, with q a final state. Being $\mathcal{R}$-related, q and s have equal encodings; because definitions are interpreted equirecursively, the unrollings of those encodings are also equal. It follows that $\hat{F}(q) = \hat{F}(s)$, and so s is also a final state.

2. We would like to prove that $q \sim\!\xrightarrow{a}\!\sim q'$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'$, or, more generally, that $q \sim\!\xrightarrow{w}\!\sim q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$. Because bisimilar states have equal encodings (part 1) and bisimilarity is reflexive (??), it suffices to show two stronger statements: (a) that $q \xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$; and (b) that $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{w}\!\sim q'$. We prove these in turn.

   (a) We shall prove that $q \xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ by induction over the structure of word w.

   - Consider the case of the empty word, ϵ; we must show that $q = q'$ implies $\hat{q} \Longrightarrow \hat{q}'$. Because the encoding is a function, this is immediate.
   - Consider the case of a nonempty word, $a\,w$; we must show that $q \xrightarrow{a}\!\xrightarrow{w} q'$ implies $w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow \hat{q}'$. Let q'_a be an a-successor of state q that is itself w-succeeded by state q'. There exists, by definition of the encoding, a trace

$$w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow w^{\mathrm{R}}\,a\,(a \setminus \hat{q}'_a) \longrightarrow w^{\mathrm{R}}\,\hat{q}'_a \Longrightarrow \hat{q}'\,,$$

     with the trace's tail justified by an appeal to the inductive hypothesis.

   (b) We shall prove that $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{w}\!\sim q'$ by induction over the structure of word w.

   - Consider the case of the empty word, ϵ; we must show that $\hat{q} \Longrightarrow \hat{q}'$ implies $q \sim q'$. By lemma 5.30, $\hat{q} \Longrightarrow \hat{q}'$ implies that q and q' have equal encodings. Part 1 can then be used to establish that q and q' are bisimilar.
   - Consider the case of a nonempty word, $a\,w$; we must show that $w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow \hat{q}'$ implies $q \xrightarrow{a}\!\xrightarrow{w}\!\sim q'$. By inversion[49], the given trace can only begin by inputting a:

$$w^{\mathrm{R}}\,a\,\hat{q} \Longrightarrow w^{\mathrm{R}}\,a\,(a \setminus \hat{q}'_a) \longrightarrow w^{\mathrm{R}}\,\hat{q}'_a \Longrightarrow \hat{q}'\,,$$

     where q'_a is an a-successor of state q. An appeal to the inductive hypothesis on the trace's tail yields $q'_a \xrightarrow{w}\!\sim q'$, and so the DFA admits $q \xrightarrow{a}\!\xrightarrow{w}\!\sim q'$, as required.

3. We shall prove that the final states are exactly those states q such that $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.

[49] Is this enough justification?

- Let q be a final state; accordingly, $\hat{F}(q) = \mathbf{1}$. There exists, by definition of the encoding, a trace

$$\epsilon\,\hat{q} \Longrightarrow \epsilon\,(\epsilon \setminus \hat{F}(q)) \longrightarrow \hat{F}(q) = \mathbf{1}\,.$$

- Assume that a trace $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$ exists. By inversion[50], this trace can only begin by inputting ϵ:

$$\epsilon\,\hat{q} \Longrightarrow \epsilon\,(\epsilon \setminus \hat{F}(q)) \longrightarrow \hat{F}(q) \Longrightarrow \mathbf{1}\,.$$

  The tail of this trace, $\hat{F}(q) \Longrightarrow \mathbf{1}$, can exist only if q is a final state. □

[50] Is this enough justification?

5.15.2 *Encoding nondeterministic finite automata?*

We would certainly be remiss if we did not attempt to generalize the rewriting specification of DFAs to one for their nondeterministic cousins.

Differently from DFA states, an NFA state q may have several nondeterministic successors for each input symbol a. To encode the NFA state q, all of its a-successors are collected in an alternative conjunction underneath the left-handed input of a. Thus, the encoding of an NFA state q becomes

$$\hat{q} \triangleq \left(\&_{a\in\Sigma}\left(a \setminus \left(\&_{q'_a}\,\hat{q}'_a\right)\right)\right) \,\&\, \left(\epsilon \setminus \hat{F}(q)\right),$$

where $\hat{F}(q)$ is defined as for DFAs.

The adjacent figure recalls from chapter 2 an NFA that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b. Using the above encoding of NFAs, ordered rewriting does indeed simulate this NFA. For example, just as there are transitions $q_0 \xrightarrow{b} q_0$ and $q_0 \xrightarrow{b} q_1$, there are traces

$$b\,\hat{q}_0 \Longrightarrow b\,\left(b \setminus (\hat{q}_0 \,\&\, \hat{q}_1)\right) \longrightarrow \hat{q}_0 \,\&\, \hat{q}_1 \begin{array}{l}\nearrow\,\hat{q}_0\\ \searrow\,\hat{q}_1\end{array}$$

Unfortunately, while it does simulate NFA behavior, this encoding is not adequate. Like DFA states, NFA states that have equal encodings are bisimilar. However, for NFAs, the converse does not hold: bisimilar states do not necessarily have equal encodings.

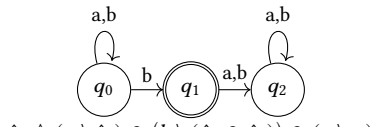


Figure 5.29: fig:ordered-rewriting:nfa-example:nfa An NFA that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b; and fig:ordered-rewriting:nfa-example:encoding its encoding

False claim 5.32. *Let $\mathcal{A} = (Q, ?, F)$ be an NFA over input alphabet Σ. Then $q \sim s$ implies $\hat{q} = \hat{s}$, for all states q and s.*

Counterexample. Consider the NFA and encoding depicted in the adjacent figure. It is easy to verify that the relation $\{q_1\} \times \{q_0, q_1\}$ is a bisimulation; in particular, q_1 simulates the $q_0 \xrightarrow{a} q_1$ transition by its self-loop, $q_1 \xrightarrow{a} q_1$. Hence, q_0 and s_0 are bisimilar. It is equally easy to verify, by unrolling the definitions used in the encoding, that $\hat{q}_0 \neq \hat{s}_0$. □

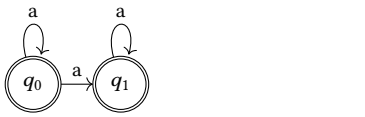


Figure 5.30: An NFA that accepts all finite words over the alphabet $\Sigma = \{a\}$

For DFAs, bisimilar states do have equal encodings because the inherent determinism DFA bisimilarity is a rather fine-grained equivalence. Because each DFA state has exactly one successor for each input symbol The additional flexibility entailed by nondeterminism

Once again, it would be possible to construct an adequate encoding, by tagging each state with a unique atom.

For the moment, we will put aside the question of an adequate encoding of NFAs.

5.15.3 *Binary representation of natural numbers*

As a further example of ordered rewriting, consider a rewriting specification of binary counters: binary representations of natural numbers equipped with increment and decrement operations.

BINARY REPRESENTATIONS In this setting, we represent a natural number in binary by an ordered context that consists of a big-endian sequence of atoms b_0 and b_1, prefixed by the atom e; leading b_0s are permitted. For example, both $\Omega = e\, b_1$ and $\Omega' = e\, b_0\, b_1$ 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. When $\Omega \approx_{\mathrm{v}} n$, we say that Ω denotes, or represents, natural number n in binary.

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

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 ordered contexts Ω that form the relation's domain of definition.[51]

These properties[52] of the $\approx_{\mathrm{v}}$ relation are proved as the following adequacy theorem.

THEOREM 5.33 (Adequacy of binary representations).

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

Surjectivity *For every natural number n, there exists a binary number Ω such that $\Omega \approx_{\mathrm{v}} n$.*

Value *If $\Omega \approx_{\mathrm{v}} n$, then $\Omega \not\longrightarrow$.*

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

Notice that the above e-v and b_0-v rules overlap when the denotation[53] is 0, giving rise to the leading b_0s that make the $\approx_{\mathrm{v}}$ relation surjective: for example, both $e\, b_1 \approx_{\mathrm{v}} 1$ and $e\, b_0\, b_1 \approx_{\mathrm{v}} 1$ hold. However, if the rule for b_0 is restricted to *nonzero* even numbers, then each natural number has a unique, canonical representation that is free of leading b_0s.[54]

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

$$\Omega ::= e \mid \Omega\, b_0 \mid \Omega\, b_1\,.$$

[52] which properties?

[53] represented natural number?

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

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

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

AN INCREMENT OPERATION To use ordered rewriting to describe an increment operation on binary representations, we introduce a new, uninterpreted atom i that will serve as an increment instruction.

Given a binary number Ω that represents n, we may append i to form a computational state, $\Omega\, i$. For i to adequately represent the increment operation, the state $\Omega\, i$ must meet two conditions, captured by the following global desiderata:

THEOREM 5.34. *Let Ω be a binary representation of n. Then:*

- some *computation from $\Omega\, i$ results in a binary representation of $n+1$ – that is, $\Omega\, i \Longrightarrow\!\approx_{\mathrm{v}} n+1$; and*
- any *computation from $\Omega\, i$ results in a binary representation of $n+1$ – that is, $\Omega\, i \Longrightarrow\!\approx_{\mathrm{v}} n'$ only if $n' = n+1$.*[55]

[55] Compare "If $\Omega\, i \Longrightarrow \Omega'$, then $\Omega' \Longrightarrow\!\approx_{\mathrm{v}} n+1$."

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 $n' = 2$.

TO IMPLEMENT THESE global desiderata locally, the previously uninterpreted atoms e, b_0, and b_1 are now given mutually recursive definitions that describe how they may be rewritten when the increment instruction, i, is encountered.

$e \triangleq e \bullet b_1 \,/\, i$ To increment e, append b_1 as a new most[56] significant bit, resulting in $e\, b_1$; the rewriting sequence $e\, i \longrightarrow e \bullet b_1 \longrightarrow e\, b_1$ is entailed by this definition.

[56] or least?

$b_0 \triangleq b_1 \,/\, i$ To increment a binary number ending in b_0, flip that bit to b_1; the entailed rewriting step is $\Omega\, b_0\, i \longrightarrow \Omega\, b_1$.

$b_1 \triangleq i \bullet b_0 \,/\, i$ 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; the entailed rewriting sequence is $\Omega\, b_1\, i \longrightarrow \Omega\, (i \bullet b_0) \longrightarrow \Omega\, i\, b_0$.

Comfortingly, $1 + 1 = 2$: that is, a computation $e\, b_1\, i \Longrightarrow e\, i\, b_0 \Longrightarrow e\, b_1\, b_0$ indeed exists.

IT SHOULD ALSO be possible to permit several increments at once, such as in $e\, b_1\, i\, i$. We could, of course, handle the increments sequentially from left to right, fully computing a binary value before moving on to the subsequent increment:

$$e\, b_1\, i\, i \Longrightarrow e\, b_1\, b_0\, i \longrightarrow e\, b_1\, b_1\,.$$

However, a strictly sequential treatment of increments would be rather disappointing. Because the ordered rewriting framework[57] is inherently concurrent, a truly concurrent treatment of multiple increments would be far more satisfying.

[57] wc?

For example, consider the several computations of $(1 + 1) + 1 = 3$ from $e\ b_1\ i\ i$:

$$e\ b_1\ i\ i \Longrightarrow e\ i\ b_0\ i \begin{array}{l} \Rightarrow\ e\ b_1\ b_0\ i \longrightarrow \\ \Rightarrow\ e\ i\ b_1 \Rightarrow \end{array} e\ b_1\ b_1$$

In other words, once the leftmost increment is carried past the least significant bit, the two increments can be processed concurrently – the increments' rewriting steps can be interleaved, with no observable difference between the various interleavings. We can even abstract from the interleavings by writing simply $e\ i\ b_0\ i \Longrightarrow e\ b_1\ b_1$.

Unfortunately, a concurrent treatment of increments falls outside the domain of **??**. Intermediate computational states, such as ...,

$$e\ i\ b_0\ i \longrightarrow e\ i\ b_1$$

because $e\ i\ b_0$ is simply not a binary value. An adequacy theorem stronger than **??** is needed.

The situation here is roughly analogous to the desire, in a functional language, for stronger metatheorems than a big-step, natural sematics admits, and we adopt a similar solution.

To this end, we define a binary relation, $\approx_{\text{I}}$, that assigns a natural number denotation to each intermediate computational state, not only to the terminal values as $\approx_{\text{V}}$ did..[58]

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

$$\frac{}{e \bullet b_1 \approx_{\text{I}} 1}\ \bullet_1\text{-I} \qquad \frac{\Omega \approx_{\text{I}} n}{\Omega\ (i \bullet b_0) \approx_{\text{I}} 2(n+1)}\ \bullet_2\text{-I}$$

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. Lastly, because the atomicity of ordered rewriting steps is very fine-grained, the $\bullet_1$-I and $\bullet_2$-I rules are needed to completely describe the valid intermediate states and their denotations. For instance, the state $e\ i$ first rewrites to the intermediate $e \bullet b_1$ before eventually rewriting to $e\ b_1$; the state $\Omega\ (i \bullet b_0)$ has a similar status.

With this $\approx_{\text{I}}$ relation in hand, we can now prove a stronger, small-step adequacy theorem.

Theorem 5.35 (Small-step adequacy of increments).
Value soundness *If $\Omega \approx_{\text{V}} n$, then $\Omega \approx_{\text{I}} n$ and $\Omega \not\longrightarrow$.*
Preservation *If $\Omega \approx_{\text{I}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\text{I}} n$.*
Progress *If $\Omega \approx_{\text{I}} n$, then either: $\Omega \longrightarrow \Omega'$ for some Ω'; or; $\Omega \approx_{\text{V}} n$.*[59]
Termination *If $\Omega \approx_{\text{I}} n$, then every rewriting sequence from Ω is finite.*

[58] 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:

$\Omega ::= e \mid \Omega\ b_0 \mid \Omega\ b_1 \mid \Omega\ i \mid e \bullet b_1 \mid \Omega\ (i \bullet b_0)$

[59] Compare with "If $\Omega \approx_{\text{I}} n$, then $\Omega \approx_{\text{V}} n$ if, and only if, $\Omega \not\longrightarrow$."

Proof. Each part is proved separately.

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

Preservation and progress can likewise be proved by structural induction on the derivation of $\Omega \approx_{\mathrm{i}} n$. In particular, the $e \bullet b_1$ and $\Omega\ (i \bullet b_0)$ rules

Termination can be proved using an explicit termination measure, $|\Omega|$, that is strictly decreasing across each rewriting, $\Omega \longrightarrow \Omega'$. Specifically, we use a measure (see the adjacent figure), adapted from the standard amortized work analysis of increment for binary counters,[60] for which $\Omega \longrightarrow \Omega'$ implies $|\Omega| > |\Omega'|$. Because the measure is always nonnegative, only finitely many such rewritings can occur.

As an example case, consider the intermediate state $\Omega\ b_0\ i$ and its rewriting $\Omega\ b_0\ i \longrightarrow \Omega\ b_1$. It follows that $|\Omega\ b_0\ i| = |\Omega| + 4 > |\Omega| + 2 = |\Omega\ b_1|$. □

$$
\begin{aligned}
|e| &= 0 & |e \bullet b_1| &= 3 \\
|\Omega\ b_0| &= |\Omega| & |\Omega\ (i \bullet b_0)| &= |\Omega| + 5 \\
|\Omega\ b_1| &= |\Omega| + 2 \\
|\Omega\ i| &= |\Omega| + 4
\end{aligned}
$$

Figure 5.31: A termination measure, adapted from the standard amortized work analysis of increment for binary counters

[60] ??.

COROLLARY 5.36 (Big-step adequacy of increments).

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

Preservation *If* $\Omega \approx_{\mathrm{i}} n$ *and* $\Omega \Longrightarrow \Omega'$, *then* $\Omega' \approx_{\mathrm{i}} n$. *In particular, if* $\Omega \approx_{\mathrm{v}} n$ *and* $\Omega\ i \Longrightarrow \approx_{\mathrm{v}} n'$, *then* $n' = n + 1$.

Proof. The two parts are proved separately.

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

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

BUT, OF COURSE, a few isolated examples do not make a proof.

By analogy with functional programming, the above adequacy conditions can be seen as stating evaluation and termination results for a big-step, evaluation semantics of increments, with $\Omega \approx_{\mathrm{v}} n$ acting as a kind of typing judgment – admittedly, a very precise one.

In functional programming, big-step results like these are usually proved by first providing a small-step operational semantics, then characterizing the valid intermediate states that arise with small steps, and finally establishing type preservation, progress, and termination results for the small-step semantics. We will adopt the same proof strategy here.

In this case, the small-step operational semantics already exists – it is simply the individual rewriting steps entailed by the definitions of e, b_0, and b_1. So our first task is to characterize the valid intermediate states that arise during a computation. To this end, we define a binary relation, $\approx_{\mathrm{i}}$, that, like the $\approx_{\mathrm{v}}$ relation, serves the dual purposes of enumerating the valid intermediate states and assigning to each state a natural number denotation.[61]

[61] As with values, we could also choose to enumerate the valid immediate states more explicitly and syntactically with a grammar:

$\Omega ::= e \mid \Omega\ b_0 \mid \Omega\ b_1 \mid \Omega\ i \mid e \bullet b_1 \mid \Omega\ (i \bullet b_0)$

$$\frac{}{e \approx_{\mathrm{I}} 0} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_0 \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_1 \approx_{\mathrm{I}} 2n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, i \approx_{\mathrm{I}} n+1}$$

$$\frac{}{e \bullet b_1 \approx_{\mathrm{I}} 1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, (i \bullet b_0) \approx_{\mathrm{I}} 2(n+1)}$$

Binary values should themselves be valid, terminal computational states, so the first three rules are carried over from the $\approx_{\mathrm{v}}$ relation. The fourth rule, involving i, allows multiple increments to be interspersed throughout the counter.

Because ordered rewriting steps are quite fine-grained, two final rules are needed to completely describe the valid intermediate states and their denotations. For instance, the state $e\, i$ first rewrites to $e \bullet b_1$ before eventually rewriting to $e\, b_1$.

Having characterized the valid intermediate states, we may state and prove the small-step adequacy of increments: preservation, progress, and termination.

THEOREM 5.37 (Small-step adequacy of increments).
Value inclusion *If* $\Omega \approx_{\mathrm{v}} n$, *then* $\Omega \approx_{\mathrm{I}} n$.
Preservation *If* $\Omega \approx_{\mathrm{I}} n$ *and* $\Omega \longrightarrow \Omega'$, *then* $\Omega' \approx_{\mathrm{I}} n$.
Progress *If* $\Omega \approx_{\mathrm{I}} n$, *then either:* $\Omega \longrightarrow \Omega'$ *for some* Ω'; *or;* $\Omega \not\longrightarrow$ *and* $\Omega \approx_{\mathrm{v}} n$.
Termination *If* $\Omega \approx_{\mathrm{I}} n$, *then every rewriting sequence from* Ω *is finite.*

Proof. Each part is proved separately.

Value inclusion can be proved by structural induction on the derivation of $\Omega \approx_{\mathrm{v}} n$.

Preservation and progress can likewise be proved by structural induction on the derivation of $\Omega \approx_{\mathrm{I}} n$. In particular, the $e \bullet b_1$ and $\Omega\, (i \bullet b_0)$ rules

Termination can be proved using an explicit termination measure, $|\Omega|$, that is strictly decreasing across each rewriting, $\Omega \longrightarrow \Omega'$. Specifically, we use a measure (see the adjacent figure), adapted from the standard amortized work analysis of increment for binary counters,[62] for which $\Omega \longrightarrow \Omega'$ implies $|\Omega| > |\Omega'|$. Because the measure is always nonnegative, only finitely many such rewritings can occur.

As an example case, consider the intermediate state $\Omega\, b_0\, i$ and its rewriting $\Omega\, b_0\, i \longrightarrow \Omega\, b_1$. It follows that $|\Omega\, b_0\, i| = |\Omega| + 4 > |\Omega| + 2 = |\Omega\, b_1|$. □

$$|e| = 0 \qquad |e \bullet b_1| = 3$$
$$|\Omega\, b_0| = |\Omega| \qquad |\Omega\, (i \bullet b_0)| = |\Omega| + 5$$
$$|\Omega\, b_1| = |\Omega| + 2$$
$$|\Omega\, i| = |\Omega| + 4$$

Figure 5.32: A termination measure, adapted from the standard amortized work analysis of increment for binary counters

[62] ??.

THEOREM 5.38 (Big-step adequacy of increments).
Preservation *If* $\Omega \approx_{\mathrm{I}} n$ *and* $\Omega \Longrightarrow \approx_{\mathrm{v}} n'$, *then* $n' = n$.
Termination? *If* $\Omega \approx_{\mathrm{I}} n$, *then* $\Omega \Longrightarrow \approx_{\mathrm{v}} n$.

Proof. Both parts are consequences of the small-step adequacy of increments (??).

Preservation is proved by structural induction on the given rewriting sequence.

The base case follows [...] by an inner structural induction on the deriva-

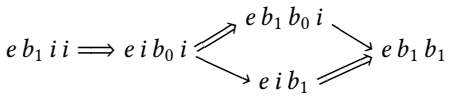
tion of $\Omega \approx_{\mathrm{v}} n'$. The inductive case can be proved by first appealing to small-step preservation (??) and then to the inductive hypothesis.

Termination? is proved by repeatedly appealing to small-step progress (??).

The small-step termination [...] (??) ensures that a value will be reached after finitely many such appeals. □

COROLLARY 5.39 (Structural adequacy of increments). *If* $\Omega \approx_{\mathrm{v}} n$, *then* $\Omega\, i \Longrightarrow \approx_{\mathrm{v}} n'$ *if, and only if,* $n' = n + 1$.

As an example computation, consider incrementing $e\, b_1$ twice, as captured by the state $e\, b_1\, i\, i$.



First, processing of the leftmost increment begins: the least significant bit is flipped, and the increment is carried over to the more significant bits. This corresponds to the reduction $e\, b_1\, i\, i \Longrightarrow e\, i\, b_0\, i$. Next, either of the two remaining increments may be processed – that is, either $e\, i\, b_0\, i \Longrightarrow e\, b_1\, b_0\, i$ or $e\, i\, b_0\, i \Longrightarrow e\, i\, b_1$.

We should like to prove the correctness of this specification of increments by establishing a computational adequacy result:

THEOREM 5.40 (Adequacy of increments). *If* $\Omega \approx_{\mathrm{v}} n$ *and* $\Omega\, i \Longrightarrow \approx_{\mathrm{v}} n'$, *then* $n' = n + 1$. *Moreover, if* $\Omega \approx_{\mathrm{v}} n$, *then* $\Omega\, i \Longrightarrow \approx_{\mathrm{v}} n + 1$.

By analogy with functional programming, this theorem can be seen as stating evaluation and termination results for a big-step evaluation semantics of increments – the judgment $\Omega \approx_{\mathrm{v}} n$ is acting as a kind of typing judgment, with n being the "type" [abstract interpretation?] of the counter Ω.

In functional programming, these sorts of big-step results are proved by first providing a small-step operational semantics, then characterizing the valid intermediate states that arise with small steps, and finally establishing type preservation, progress, and termination results for the small-step semantics. We will adopt the same strategy here.

First, we define a relation, $\approx_{\mathrm{I}}$, that characterizes the valid intermediate states that arise during increments.

To prove this ??, we will first introduce an auxiliary relation, $\approx_{\mathrm{I}}$, that characterizes the valid states that arise during increments. This relation is defined inductively by the following rules.

$$\frac{}{e \approx_{\mathrm{I}} 0} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_0 \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_1 \approx_{\mathrm{I}} 2n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, i \approx_{\mathrm{I}} n+1}$$

$$\frac{}{e \bullet b_1 \approx_{\mathrm{I}} 1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, (i \bullet b_0) \approx_{\mathrm{I}} 2(n+1)}$$

The latter two

A DECREMENT OPERATION Binary counters may also be equipped with a decrement operation. Instead of examining decrements *per se*, we will implement a closely related operation: the normalization of binary representations to what might be called *head-unary form.*[63] An ordered context Ω will be said to be in head-unary form if it has one of two forms: $\Omega = z$; or $\Omega = \Omega' \; s$, for some binary number Ω'.

[63] We will frequently abuse terminology, using 'head-unary normalization' and 'decrement' interchangably.

Just as appending the atom i to a counter initiates an increment, appending an uninterpreted atom d will cause the counter to begin normalizing to head-unary form. The following ?? serves as a specification of head-unary normalization, relating a value's head-unary form to its denotation.

THEOREM 5.41 (Structural adequacy of decrements). *If* $\Omega \approx_{\mathrm{v}} n$, *then:*

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

For example, because $e \; b_1$ denotes 1, a computation $e \; b_1 \; d \Longrightarrow \Omega' \; s$ must exist, for some $\Omega' \approx_{\mathrm{v}} 0$.

ONCE AGAIN, to implement these desiderata locally, the recursive definitions of e, b_0, and b_1 will be revised with an additional clause that handles decrements; also, a recursively defined proposition b_0' is introduced:

$e \triangleq (\cdots \; / \; i) \; \& \; (z \; / \; d)$ Because e denotes 0, its head-unary form is simply z.

$b_0 \triangleq (\cdots \; / \; i) \; \& \; (d \bullet b_0' \; / \; d)$ Because $\Omega \; b_0$ denotes $2n$ if Ω denotes n, its head-unary form can be contructed by recursively putting the more significant bits into head-unary form and appending b_0' to process that result.

$b_0' \triangleq (z \setminus z) \; \& \; (s \setminus b_1 \bullet s)$ If the more significant bits have head-unary form z and therefore denote 0, then $\Omega \; b_0$ also denotes 0 and has head-unary form z. Otherwise, if they have head-unary form $\Omega' \; s$ and therefore denote $n > 0$, then $\Omega \; b_0$ denotes $2n$ and has head-unary form $\Omega' \; b_1 \; s$, which can be constructed by replacing s with $b_1 \; s$.

$b_1 \triangleq (\cdots \; / \; i) \; \& \; (b_0 \bullet s \; / \; d)$ Because $\Omega \; b_1$ denotes $2n + 1$ if Ω denotes n, its head-unary form, $\Omega \; b_0 \; s$, can be constructed by flipping the least significant bit to b_0 and appending s.

Comfortingly, $2 - 1 = 1$: the head-unary form of $e \; b_1$ is $e \; b_0 \; b_1 \; s$:

$$e \; b_1 \; b_0 \; d \Longrightarrow e \; b_1 \; d \; b_0' \Longrightarrow e \; b_0 \; s \; b_0' \Longrightarrow e \; b_0 \; b_1 \; s \; .$$

AT THIS POINT, we would like to prove the adequacy of decrements. However, having just revised the definitions of e, b_0, and b_1, we must first recheck the adequacy of binary representation(see ??). Unfortunately, the newly introduced alternative conjunctions, together with the fine-grained atomicity of ordered rewriting, cause [...].

FALSE CLAIM 5.42 (Adequacy of binary representations).
Functional *For every binary number Ω, there exists a unique natural number n such that $\Omega \approx_{\mathrm{v}} n$.*
Surjectivity *For every natural number n, there exists a binary number Ω such that $\Omega \approx_{\mathrm{v}} n$.*
Values *If $\Omega \approx_{\mathrm{v}} n$, then $\Omega \not\longrightarrow$.*

Counterexample. Although the $\approx_{\mathrm{v}}$ relation remains functional and surjective, it does not satisfy [...]. Because $e \approx_{\mathrm{v}} 0$, the counter e is a value (with denotation 0). However, because the atomicity of ordered rewriting is extremely fine-grained, e can be rewritten:

$$e = (e \bullet b_1 \mathbin{/} i) \mathbin{\&} (z \mathbin{/} d) \begin{array}{l} \nearrow\ e \bullet b_1 \mathbin{/} i \\ \searrow\ z \mathbin{/} d \end{array}$$

That e is an active proposition violates our conception of values as inactive. □

AT THIS POINT, we would like to prove the adequacy of decrements. However, having just revised the definitions of e, b_0, and b_1, we must first recheck the adequacy of increments. Unfortunately, the newly introduced alternative conjunctions, together with the fine-grained atomicity of ordered rewriting, cause the preservation and progress properties to fail.

FALSE CLAIM 5.43 (Small-step adequacy of increments).
Value inclusion *If $\Omega \approx_{\mathrm{v}} n$, then $\Omega \approx_{\mathrm{i}} n$.*
Preservation *If $\Omega \approx_{\mathrm{i}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\mathrm{i}} n$.*
Progress *If $\Omega \approx_{\mathrm{i}} n$, then either: $\Omega \longrightarrow \Omega'$ for some Ω'; $\Omega \not\longrightarrow$ and $\Omega \approx_{\mathrm{v}} n$*
Termination *If $\Omega \approx_{\mathrm{i}} n$, then every rewriting sequence from Ω is finite.*

Counterexample. As a counterexample to preservation, notice that $e\,i$ denotes 1 and that

$$e\,i = (e \bullet b_1 \mathbin{/} i) \mathbin{\&} (z \mathbin{/} d) \longrightarrow (e \bullet b_1 \mathbin{/} i)\,i\,,$$

but that $(e \bullet b_1 \mathbin{/} i)\,i$ does not have a denotation under the $\approx_{\mathrm{i}}$ relation.

Even worse, computations can now enter stuck states – $e\,i \longrightarrow (z/d)\,i \not\longrightarrow$, for example. It's difficult to imagine assigning denotations to these stuck states, making them counterexamples to preservation. Even if denotations were somehow assigned to them, such states would anyway violate the desired progress theorem. □

In both cases, these counterexamples arise from the very fine-grained atomicity of ordered rewriting. Now that the definitions of e, b_0, and b_1 include alternative conjunctions, [...].

THEOREM 5.44. *Evaluation* *If $\Omega \approx_{\mathrm{i}} n$, then $\Omega \Longrightarrow \approx_{\mathrm{v}} n$. In particular, if $\Omega \approx_{\mathrm{v}} n$, then $\Omega \Longrightarrow n+1 \approx_{\mathrm{v}}$.*

Preservation *If* $\Omega \approx_{\text{I}} n$ *and* $\Omega \Longrightarrow\!\approx_{\text{V}} n'$, *then* $n' = n$.

Proof. By structural induction on the given derivation of $\Omega \approx_{\text{I}} n$. □

The solution is to chain several small rewriting steps together into a single, larger atomic step.

5.16 *Weakly focused rewriting*

Andreoli:??'s observation was that propositions can be partitioned into two classes, or *polarities*[64], according to the invertibility of their sequent calculus rules, and that [...].

[64] reference?

The ordered propositions are polarized into two classes, the positive and negative propositions, according to the invertibility of their sequent calculus rules.

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

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

The positive propositions are those propositions that have invertible left rules, such as ordered conjunction; the negative propositions are those that have invertible right rules, such as the ordered implications.

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

Left rules for negative connectives may be chained together into a single *left-focusing phase*, reflected by the pattern judgment $\Omega_L\ [A^-]\ \Omega_R \Vdash C^+$. Following **Zeilberger:??**, this judgment can be read as a function of an in-focus negative proposition, A^-, that produces the ordered contexts Ω_L and Ω_R and the positive consequent C^+ as outputs.

The left-focus judgment is defined inductively on the structure of the in-focus proposition by the following rules.

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

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

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

These rules parallel the usual sequent calculus rules, maintaining focus on the subformulas of negative polarity. First, the $\uparrow$L rule finishes a left-focusing phase by producing the consequent A^+ from $\uparrow A^+$.

Second, the $\setminus$L′ and /L′ rules diverge slightly from the usual left rules for left- and right-handed implication in that they have no premises decomposing [the antecedent[65]] A^+. This would mean that a weakly focused sequent

[65] wc?

calculus based on $\backslash\text{L}'$ and $/\text{L}'$ would be incomplete for provability. It is possible to [...].[66] However, because our goal here is a rewriting framework and such a framework is inherently incomplete[67], [...].

[66] **Simmons:CMU??.**

[67] Is this right?

$$\frac{\Omega_L\ [A^-]\ \Omega_R \Vdash C^+}{\Omega_L \downarrow A^-\ \Omega_R \longrightarrow C^+}\ \downarrow\text{D}$$

Consider the recursively defined proposition $\alpha \triangleq (\beta \setminus \alpha) \mathbin{\&} (\gamma \setminus \mathbf{1})$. Previously, in the unfocused rewriting framework, it took two steps to rewrite $\beta\,\alpha$ into α:

$$\beta\,\alpha = \beta\,\big((\beta \setminus \alpha) \mathbin{\&} (\gamma \setminus \mathbf{1})\big) \longrightarrow \beta\,(\beta \setminus \alpha) \longrightarrow \alpha$$

Now, in the polarized, weakly focused rewriting framework, the analogous recursive definition is $\alpha^- \triangleq (\beta^+ \setminus \uparrow\downarrow\alpha^-) \mathbin{\&} (\gamma^+ \setminus \uparrow\mathbf{1})$, and it takes only one step to rewrite $\beta^+ \downarrow\alpha^-$ into $\downarrow\alpha^-$:

$$\beta^+ \downarrow\alpha^- = \beta^+ \downarrow\big((\beta^+ \setminus \uparrow\downarrow\alpha^-) \mathbin{\&} (\gamma^+ \setminus \uparrow\mathbf{1})\big) \longrightarrow \downarrow\alpha^-$$

because $\beta^+\ [\alpha^-] \Vdash \downarrow\alpha^-$.

Notice that, because the left-focus judgment is defined inductively, there are some recursively defined negative propositions that cannot successfully be put in focus. For example, under the definition $\alpha^- \triangleq \beta^+ \setminus \alpha^-$, there are no contexts Ω_L and Ω_R and conseqeunt C^+ for which $\Omega_L\ [\alpha^-]\ \Omega_R \Vdash C^+$ is derivable.

In addition to the $\downarrow\text{D}$ rule for decomposing $\downarrow A^-$, weakly focused ordered rewriting retains the $\bullet\text{D}$ and $\mathbf{1}\text{D}$ rules for decomposing $A^+ \bullet B^+$ and $\mathbf{1}$ and the compatability rules, $\longrightarrow\text{C}_\text{L}$ and $\longrightarrow\text{C}_\text{R}$. Together, these five rules and the left focus[68] rules comprise the weakly focused ordered rewriting framework; they are summarized in **??**.

[68] focal?

Weakly focused ordered rewriting is sound with respect to the unfocused rewriting framework of **??**. Given a depolarization function $(-)^\circ$ that maps polarized propositions and contexts to their unpolarized counterparts,we may state and prove the following soundness theorem for weakly focused rewriting.

$$(\Omega_1\ \Omega_2)^\circ = \Omega_1^\circ\ \Omega_2^\circ \qquad (\downarrow A^-)^\circ = (A^-)^\circ$$
$$(\cdot)^\circ = \cdot \qquad (\uparrow A^+)^\circ = (A^+)^\circ$$
$$(A^+)^\circ = (A^+)^\circ \qquad (A^+ \setminus B^-)^\circ = (A^+)^\circ \setminus (B^-)^\circ$$
$$\textit{etc.}$$

Figure 5.34: Depolarization of propositions and contexts

Theorem 5.45 (Soundness of weakly focused rewriting). *If* $\Omega \Longrightarrow \Omega'$, *then* $\Omega^\circ \Longrightarrow (\Omega')^\circ$.

Proof. By structural induction on the given rewriting step, after generalizing the inductive hypothesis to include:

- If $\Omega \longrightarrow \Omega'$, then $\Omega^\circ \Longrightarrow (\Omega')^\circ$.
- If $\Omega_L\ [A^-]\ \Omega_R \Vdash C^+$, then $(\Omega_L \downarrow A^-\ \Omega_R)^\circ \Longrightarrow (C^+)^\circ$. □

A completeness theorem also holds, but we forgo its development because it is not essential to the remainder of this work.

Second, with the lone exception negative propositions are latent[69] –

[69] **??.**

Figure 5.33: A weakly focused ordered rewriting framework

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

NEGATIVE PROPS. $A^- ::= \alpha^- \mid A^+ \setminus B^- \mid B^- / A^+ \mid A^- \,\&\, B^- \mid \top \mid \uparrow A^+$

ORDERED CONTEXTS $\Omega ::= \Omega_1\, \Omega_2 \mid \cdot \mid A^+$

REWRITING: $\Omega \longrightarrow \Omega'$ AND $\Omega \Longrightarrow \Omega'$

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

(no $\oplus$D and $\mathbf{0}$D rules)

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\, \Omega_2 \longrightarrow \Omega_1'\, \Omega_2}\ \longrightarrow\!\text{C}_\text{L} \qquad \frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\, \Omega_2 \longrightarrow \Omega_1'\, \Omega_2}\ \longrightarrow\!\text{C}_\text{R}$$

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

LEFT FOCUS: $\Omega_L\, [A^-]\, \Omega_R \Vdash C^+$

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

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

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

5.17 *Revisiting automata*

$$\hat{q} \triangleq \left(\&_{a\in\Sigma}(a \setminus \uparrow\downarrow\hat{q}'_a)\right) \,\&\, (\epsilon \setminus \uparrow\hat{F}(q))$$

where

$$q \xrightarrow{a} q'_a, \text{ for all } a \in \Sigma \quad \text{and} \quad \hat{F}(q) = \begin{cases} \mathbf{1} & \text{if } q \in F \\ \downarrow\top & \text{if } q \notin F \end{cases}$$

THEOREM 5.46 (DFA adequacy up to bisimilarity). *Let $\mathcal{A} = (Q, ?, F)$ be a DFA over the input alphabet Σ. Then, for all states q, q', and s:*

1. *$q \sim s$ if, and only if, $\hat{q} = \hat{s}$.*
2. *$q \sim\!\xrightarrow{a}\!\sim q'$ if, and only if, $a\,\hat{q} \Longrightarrow \hat{q}'$, for all input symbols $a \in \Sigma$. More generally, $q \sim\!\xrightarrow{w}\!\sim q'$ if, and only if, $w^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$, for all finite words $w \in \Sigma^*$.*
3. *$q \in F$ if, and only if, $\epsilon\,\hat{q} \Longrightarrow \mathbf{1}$.*

Lemma?? is still needed, but now has a much different proof. Previously, the proof of ?? relied on a very specific and delicate property of DFAs, namely that each DFA state has one and only one a-successor for each input symbol a. Now, with weakly focused ordered rewriting, the ??'s proof is much less fragile. With the larger granularity of individual rewriting steps that the weakly focused framework affords, a state's encoding is a latent proposition

5.18 *Revisiting binary counters*

With ordered rewriting now based on a weakly focused sequent calculus, we can revisit our previous attempt to extend binary counters with support for decrements or head-unary normalization.

The propositions e, b_0, b'_0, and b_1 are recursively defined in nearly the same way as before. With one exception discussed below, only the necessary shifts are inserted to consistently assign a negative polarity to the defined atoms e, b_0, b'_0, and b_1 and a positive polarity to the uninterpreted atoms i, d, z, and s.

$$\begin{aligned} e &\triangleq (e \bullet b_1 \,/\, i) \,\&\, (z \,/\, d) \\ b_0 &\triangleq (\uparrow\downarrow b_1 \,/\, i) \,\&\, (d \bullet b'_0 \,/\, d) \\ b'_0 &\triangleq (z \setminus z) \,\&\, (s \setminus b_1 \bullet s) \\ b_1 &\triangleq (i \bullet b_0 \,/\, i) \,\&\, (b_0 \bullet s \,/\, d) \end{aligned}$$

VALUES Once again, we use the same $\approx_{\mathrm{v}}$ relation to assign a unique natural number denotation to each binary representation.

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

Because the underlying ordered rewriting framework has changed, we must verify that $\approx_{\mathrm{v}}$ is adequate – inparticular, the [...] property that values cannot be independently rewritten.

THEOREM 5.47 (Adequacy of binary representations).
Functional *For every binary number Ω, there exists a unique natural number n such that $\Omega \approx_{\mathrm{V}} n$.*
Surjectivity *For every natural number n, there exists a binary number Ω such that $\Omega \approx_{\mathrm{V}} n$.*
Value *If $\Omega \approx_{\mathrm{V}} n$, then $\Omega \not\longrightarrow$.*

Proof. By induction over the structure of Ω. As an example, consider the case in which $e \approx_{\mathrm{V}} 0$. Indeed, $e \not\longrightarrow$ because $e = (e \bullet b_1 \,/\, i) \,\&\, (z \,/\, d)$ and

$\Omega_L\ [(e \bullet b_1 \,/\, i) \,\&\, (z \,/\, d)]\ \Omega_R \Vdash C^+$ only if $\Omega_L = \cdot$ and either $\Omega_R = i$ or $\Omega_R = d$.

The other cases are similar. □

INCREMENT Previously, under the unfocused rewriting framework[70], rewriting $e\ i$ into $e \bullet b_1$ took two small steps:

$$e\ i = \big((e \bullet b_1 \,/\, i) \,\&\, (z \,/\, d)\big)\ i \longrightarrow (e \bullet b_1 \,/\, i)\ i \longrightarrow e \bullet b_1$$

But now, with weakly focused rewriting, those two steps are combined into one atomic whole: $e\ i \longrightarrow e \bullet b_1$.

As for the unfocused rewriting implementation of binary increments, we use a $\approx_{\mathrm{I}}$ relation to assign a natural number denotation to each computational state. In fact, the specific definition of the $\approx_{\mathrm{I}}$ remains unchanged from ??:

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

$$\frac{}{e \bullet b_1 \approx_{\mathrm{I}} 1}\ \bullet_1\text{-I} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\ (i \bullet b_0) \approx_{\mathrm{I}} 2(n+1)}\ \bullet_2\text{-I}$$

The only exception to [...] is the appearance of $\uparrow\downarrow b_1$ in the definition of b_0. Without this double shift, $e\ b_0\ i$ would be latent, unable to rewrite to a value until a second increment is appended, because the necessary $[(b_1 \,/\, i) \,\&\, (d \bullet b_0' \,/\, d)]\ i \Vdash b_1$ is not derivable. However, with the double shift, $e\ b_0\ i \longrightarrow e\ b_1$ because $[(\uparrow\downarrow b_1 \,/\, i) \,\&\, (d \bullet b_0' \,/\, d)]\ i \Vdash \downarrow b_1$ is derivable.

With weakly focused rewriting, it is no longer possible to reach the stuck state ....

THEOREM 5.48 (Small-step adequacy of increments).
Value soundness *If $\Omega \approx_{\mathrm{V}} n$, then $\Omega \approx_{\mathrm{I}} n$ and $\Omega \not\longrightarrow$.*
Preservation *If $\Omega \approx_{\mathrm{I}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\mathrm{I}} n$.*
Progress *If $\Omega \approx_{\mathrm{I}} n$, then either: $\Omega \longrightarrow \Omega'$ for some Ω'; or; $\Omega \approx_{\mathrm{V}} n$.*[71]
Termination *If $\Omega \approx_{\mathrm{I}} n$, then every rewriting sequence from Ω is finite.*

Proof. As before, each part is proved separately.

Value soundness, preservation, and progress can be proved by structural induction on the derivation of $\Omega \approx_{\mathrm{I}} n$.

Termination can be proved using the same explicit termination measure, $|\Omega|$, as in ??. □

[70] wc?

[71] Compare with "If $\Omega \approx_{\mathrm{I}} n$, then $\Omega \approx_{\mathrm{V}} n$ if, and only if, $\Omega \not\longrightarrow$."

DECREMENTS

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

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(d \bullet b_0') \approx_{\mathrm{D}} 2n}\ \bullet_1\text{-D}$$

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_0 \bullet s) \approx_{\mathrm{D}} 2n+1}\ \bullet_2\text{-D} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_1 \bullet s) \approx_{\mathrm{D}} 2n+2}\ \bullet_3\text{-D}$$

THEOREM 5.49 (Small-step adequacy of decrements).
Preservation *If* $\Omega \approx_{\mathrm{D}} n$ *and* $\Omega \longrightarrow \Omega'$, *then* $\Omega' \approx_{\mathrm{D}} n$.
Progress *If* $\Omega \approx_{\mathrm{D}} n$, *then [either]:*

- $\Omega \longrightarrow \Omega'$, *for some* Ω';
- $n = 0$ *and* $\Omega = z$; *or*
- $n > 0$ *and* $\Omega = \Omega'\, s$, *for some* Ω' *such that* $\Omega' \approx_{\mathrm{I}} n - 1$.

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

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

Termination is proved by exhibiting a measure, $|-|_{\mathrm{D}}$, that is strictly decreasing across each rewriting. Following the example of termination for increment-only binary counters(??), we could try to assign a constant amount of potential to each of the counter's constituents. Leaving these potentials as unknowns, we can generate a set of constraints from the allowed rewritings and then attempt to solve them.

For instance, here are several rewritings and their corresponding potential constraints.

Some selected rewritings	*Potential constraints*
$\Omega\, b_1\, i \longrightarrow \Omega\,(i \bullet b_0) \longrightarrow \Omega\, i\, b_0$	$b_1 + i > i + b_0 + 1$
$\Omega\, b_0\, d \longrightarrow \Omega\,(d \bullet b_0') \longrightarrow \Omega\, d\, b_0'$	$b_0 + d > d + b_0' + 1$
$\Omega\, s\, b_0' \longrightarrow \Omega\,(b_1 \bullet s) \longrightarrow \Omega\, b_1\, s$	$s + b_0' > b_1 + s + 1$

These constraints are satisfiable only if $b_1 > b_0 > b_0' > b_1$, which is, of course, impossible.

However, notice that each b_1 that arises from an interaction between s and b_0' will never participate in further rewritings because any increments remaining to the left of b_1 will only involve more significant bits, not this less significant b_1. A similar argument can be made for all bits that occur between the rightmost i and the terminal s, suggesting that those bits be assigned no potential at all.

This leads to the termination measure, $|-|_{\mathrm{D}}$, and its auxiliary measures, $|-|_{\mathrm{I}}$ and $|-|_{\mathrm{s}}$, shown in the adjacent ??. (Note that the measure $|-|_{\mathrm{I}}$ is not the same as the measure used for increment-only binary counters(??).)x

is proved by exhibiting a pair of measures, $|\Omega|_d$ and $|\Omega|_s$, ordered lexicographically:

- If $\Omega \approx_{\mathrm{D}} n$ and $\Omega \longrightarrow \Omega'$, then either: $|\Omega|_d > |\Omega'|_d$; or $|\Omega|_d = 0$ and $|\Omega|_s > |\Omega'|_s$.

These measures are shown in the adjacent ??, rely on an auxiliary measure, $|\Omega|$, for increment states. Unfortunately, it is not possible to simply reuse the measure from ??. In that measure, each b_0 bit was assigned no potential. With decrements, however, b_0 needs to carry enough potential to transfer to b_0' in case a decrement instruction is encountered.

$$
\begin{array}{ll}
|\Omega\, s|_s = |\Omega| & |\Omega\, d|_d = |\Omega| + 1 \\
|e| = 0 & |\Omega\, b_0'|_d = |\Omega|_d + 2 \\
|\Omega\, b_0| = |\Omega| + 4 & |z|_d = 0 \\
|\Omega\, b_1| = |\Omega| + 6 & |\Omega\, s|_d = 0 \\
|\Omega\, i| = |\Omega| + 8 & |\Omega\, (d \bullet b_0')|_d = |\Omega| + 4 \\
|e \bullet b_1| = 7 & |\Omega\, (b_0 \bullet s)|_d = 1 \\
|\Omega\, (i \bullet b_0)| = |\Omega| + 13 & |\Omega\, (b_1 \bullet s)|_d = 1
\end{array}
$$

For the rewritings $\Omega\, b_1\, i \longrightarrow \Omega\, (i \bullet b_0) \longrightarrow \Omega\, i\, b_0$, the assigned potentials must satisfy $b_1 + i > i + b_0 + 1$

No

As an example case, consider the intermediate state $\Omega\, (b_1 \bullet s)$ and its rewriting $\Omega\, (b_1 \bullet s) \longrightarrow \Omega\, b_1\, s$. It follows $|\Omega\, (b_1 \bullet s)|_d = 1 > 0 = |\Omega\, b_1\, s|_d$. Any subsequent rewritings of Ω are justified by a decrease in $|\Omega\, b_1\, s|_s > |\Omega|$. □

Corollary 5.50 (Big-step adequacy of decrements). *If $\Omega \approx_{\mathrm{D}} n$, then:*

- $\Omega \Longrightarrow z$ *if, and only if,* $n = 0$*;*
- $\Omega \Longrightarrow \Omega'\, s$ *for some* Ω' *such that* $\Omega' \approx_{\mathrm{I}} n - 1$*, if* $n > 0$*; and*
- $\Omega \Longrightarrow \Omega'\, s$ *only if* $n > 0$ *and* $\Omega' \approx_{\mathrm{I}} n - 1$.

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

$$
\begin{array}{lll}
|\Omega\, d|_{\mathrm{D}} = |\Omega|_{\mathrm{I}} + 1 & |e|_{\mathrm{I}} = 0 & |e|_{\mathrm{S}} = |e|_{\mathrm{I}} = 0 \\
|\Omega\, b_0'|_{\mathrm{D}} = |\Omega|_{\mathrm{D}} + 2 & |\Omega\, b_0|_{\mathrm{I}} = |\Omega|_{\mathrm{I}} + 4 & |\Omega\, b_0|_{\mathrm{S}} = |\Omega|_{\mathrm{S}} \\
|z|_{\mathrm{D}} = 0 & |\Omega\, b_1|_{\mathrm{I}} = |\Omega|_{\mathrm{I}} + 6 & |\Omega\, b_1|_{\mathrm{S}} = |\Omega|_{\mathrm{S}} \\
|\Omega\, s|_{\mathrm{D}} = |\Omega|_{\mathrm{S}} & |\Omega\, i|_{\mathrm{I}} = |\Omega|_{\mathrm{I}} + 8 & |\Omega\, i|_{\mathrm{S}} = |\Omega\, i|_{\mathrm{I}} = |\Omega|_{\mathrm{I}} + 8 \\
|\Omega\, (d \bullet b_0')|_{\mathrm{D}} = |\Omega\, d\, b_0'|_{\mathrm{D}} + 1 & |e \bullet b_1|_{\mathrm{I}} = |e\, b_1|_{\mathrm{I}} + 1 & |e \bullet b_1|_{\mathrm{S}} = |e\, b_1|_{\mathrm{S}} + 1 \\
|\Omega\, (b_0 \bullet s)|_{\mathrm{D}} = |\Omega\, b_0\, s|_{\mathrm{D}} + 1 & |\Omega\, (i \bullet b_0)|_{\mathrm{I}} = |\Omega\, i\, b_0|_{\mathrm{I}} + 1 & |\Omega\, (i \bullet b_0)|_{\mathrm{S}} = |\Omega\, i\, b_0|_{\mathrm{S}} + 1 \\
|\Omega\, (b_1 \bullet s)|_{\mathrm{D}} = |\Omega\, b_1\, s|_{\mathrm{D}} + 1 & &
\end{array}
$$

- If $\Omega \approx_{\mathrm{D}} n$ and $\Omega \longrightarrow \Omega'$, then $|\Omega|_{\mathrm{D}} > |\Omega'|_{\mathrm{D}}$.
- If $\Omega \approx_{\mathrm{I}} n$ and $\Omega \longrightarrow \Omega'$, then $|\Omega|_{\mathrm{I}} > |\Omega'|_{\mathrm{I}}$ and $|\Omega|_{\mathrm{S}} > |\Omega'|_{\mathrm{S}}$.

5.19 *Temporary*

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

$$\frac{\Omega \approx_{\mathrm{D}} n}{\Omega\, (d \bullet b_0') \approx_{\mathrm{D}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, (b_1 \bullet s) \approx_{\mathrm{D}} 2n+2} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, (b_0 \bullet s) \approx_{\mathrm{D}} 2n+1}$$

As the first rule exhibits, a binary number and its head-unary form denote the same value. The last three rules are included by analogy with the $e \bullet b_1$ and $i \bullet b_0$ rules of the $\approx_{\mathrm{I}}$ relation.

False claim 5.51 (Small-step adequacy of decrements).
Preservation *If* $\Omega \approx_{\mathrm{D}} n$ *and* $\Omega \longrightarrow \Omega'$, *then* $\Omega' \approx_{\mathrm{D}} n$.
Progress *If* $\Omega \approx_{\mathrm{D}} n$, *then either:*

- $\Omega \longrightarrow \Omega'$;
- $n = 0$ *and* $\Omega = z$; *or*
- $n > 0$ *and* $\Omega = \Omega'\, s$ *for some* Ω' *such that* $\Omega' \approx_{\mathrm{I}} n - 1$.

Productivity *If* $\Omega \approx_{\mathrm{D}} n$, *then every rewriting sequence from* Ω *has a finite prefix* $\Omega \Longrightarrow \Omega'$ *such that either:*

- $n = 0$ *and* $\Omega' = z$; *or*
- $n > 0$ *and* $\Omega' = \Omega_0'\, s$, *for some* Ω_0' *such that* $\Omega_0' \approx_{\mathrm{I}} n - 1$.

Proof. The fine-grained atomicity of ordered rewriting, together with the use of alternative conjunction in the recursively defined propositions e, b_0, b_0', and b_1, causes both preservation and progress properties to fail.

As a counterexample to preservation, $e\, d \approx_{\mathrm{D}} 0$ and $e\, d \longrightarrow (z \mathbin{/} d)\, d$, but $(z \mathbin{/} d)\, d \approx_{\mathrm{D}} 0$ does *not* hold.

Even worse, the fine-grained atomicity of ordered rewriting means that computations can enter stuck states, which shouldn't have denotations and which would violate progress if they were somehow assigned denotations. For example, $e\, d \approx_{\mathrm{D}} 0$ and $e\, d \longrightarrow (e \bullet b_1 \mathbin{/} i)\, d \not\longrightarrow$.

$e\, i \approx_{\mathrm{I}} 0$ and $e\, i \longrightarrow (z \mathbin{/} d)\, i \not\longrightarrow$ □

These binary counters may also be equipped with a decrement operation. Although "decrement" is a convenient name for this operation, it is more accurate to implement decrements by converting the binary representation to what might be called *head-unary form*: an ordered context Ω is said to be in head-unary form if either: $\Omega = z$; or $\Omega = \Omega_0\, s$ for some binary representation Ω_0.

Similar to how the atom i is used to describe increments, a decrement is initiated by appending an atom d to the counter; d is then processed from right to left by the counter's bits. To support this, the definitions of e, b_0, and

b_1 are revised

$$e \triangleq (e \bullet b_1 \mathbin{/} i) \mathbin{\&} (\cdots \mathbin{/} d)$$
$$b_0 \triangleq (b_1 \mathbin{/} i) \mathbin{\&} (\cdots \mathbin{/} d)$$
$$b_1 \triangleq (i \bullet b_0 \mathbin{/} i) \mathbin{\&} (\cdots \mathbin{/} d)$$

To initiate a decrement of a counter Ω, we append the uninterpreted atom d to the counter, forming $\Omega\, d$.

To implement the decrement operation, we instead

Although "decrement" is a convenient name for this operation, it is perhaps more accurate to think of this operation as putting the binary representation into a head-unary form: either z or $\Omega'\, s$ for some $\Omega' \approx_{\mathrm{I}} n - 1$.

- If $\Omega \approx_{\mathrm{I}} n$, then:
  - $n = 0$ if, and only if, $\Omega\, d \Longrightarrow z$; and
  - $n > 0$ implies $\Omega\, d \Longrightarrow \Omega'\, s$ for some Ω' such that $\Omega' \approx_{\mathrm{I}} n - 1$; and
  - $\Omega\, d \Longrightarrow \Omega'\, s$ implies $n > 0$ and $\Omega' \approx_{\mathrm{I}} n - 1$.

$$e \triangleq (e \bullet b_1 \mathbin{/} i) \mathbin{\&} (z \mathbin{/} d)$$
$$b_0 \triangleq (b_1 \mathbin{/} i) \mathbin{\&} (d \bullet b_0' \mathbin{/} d)$$
$$b_0' \triangleq (z \setminus z) \mathbin{\&} (s \setminus b_1 \bullet s)$$
$$b_1 \triangleq (i \bullet b_0 \mathbin{/} i) \mathbin{\&} (b_0 \bullet s \mathbin{/} d)$$

$e \triangleq \cdots \mathbin{\&} (z \mathbin{/} d)$ Because the counter e represents 0, its head-unary form is simply z.

$b_1 \triangleq \cdots \mathbin{\&} (b_0 \bullet s \mathbin{/} d)$ Because the counter $\Omega\, b_1$ represents $2n{+}1 > 0$ when Ω represents n, its head-unary form must then be the successor of a counter representing $2n$ – that is, $\Omega\, b_0\, s$.

$b_0 \triangleq \cdots \mathbin{\&} (d \bullet b_0' \mathbin{/} d)$ The natural number that the counter $\Omega\, b_0$ represents could be either zero or positive, depending on whether Ω represents zero or a positive natural number. Thus, to put $\Omega\, b_0$ into head-unary form, we first put Ω into head-unary form and then use b_0' to branch on the result.

$b_0' \triangleq (z \setminus z) \mathbin{\&} (s \setminus b_1 \bullet s)$ If the head-unary form of Ω is z, then $\Omega\, b_0$ also represents 0 and has head-unary form z. Otherwise, if the head-unary form of Ω is $\Omega'\, s$ for some $\Omega' \approx_{\mathrm{I}} n'$, then $\Omega\, b_0$ represents $2n' + 2$ and has head-unary form $\Omega'\, b_1\, s$.

Decrements actually do not literally decrement the counter, but instead put it into a "head unary" form in which the couter is either z or s with a binary counter beneath.

We will use the same strategy for proving the adequacy of decrements as we did for increments: Characterize the valid states

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

$$\frac{}{e \bullet b_1 \mathbin{/} i \approx_{\mathrm{I}} 0} \qquad \frac{}{z \mathbin{/} d \approx_{\mathrm{I}} 0}$$

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_1 \mathbin{/} i) \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(d \bullet b_0' \mathbin{/} d) \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(d \bullet b_0') \approx_{\mathrm{D}} 2n}$$

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(i \bullet b_0 \mathbin{/} i) \approx_{\mathrm{I}} 2n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_0 \bullet s \mathbin{/} d) \approx_{\mathrm{I}} 2n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(i \bullet b_0) \approx_{\mathrm{I}} 2n+2} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_0 \bullet s) \approx_{\mathrm{D}} 2n+1}$$

$$\frac{\Omega \approx_{\mathrm{D}} n}{\Omega\,(z \setminus z) \approx_{\mathrm{D}} 2n} \qquad \frac{\Omega \approx_{\mathrm{D}} n}{\Omega\,(s \setminus b_1 \bullet s) \approx_{\mathrm{D}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_1 \bullet s) \approx_{\mathrm{D}} 2n+2}$$

Notice that $e\, s\, b_0' \approx_{\mathrm{D}} 0$ but $e\, s\, b_0' \Longrightarrow e\, b_1\, s \approx_{\mathrm{D}} 1$. If we revise the s rule to use $n+1$, then a different problem arises: $e\, b_1\, d \approx_{\mathrm{D}} 0$ but $e\, b_1\, d \Longrightarrow e\, b_0\, s \approx_{\mathrm{D}} 1$.

THEOREM 5.52 (Adequacy). *If $\Omega \approx_{\mathrm{I}} n$, then:*

- *$n = 0$ if and only if $\Omega\, d \Longrightarrow z$; and*
- *$n > 0$ implies $\Omega\, d \Longrightarrow \Omega'\, s$ and $\Omega' \approx_{\mathrm{I}} n-1$;*
- *$\Omega\, d \Longrightarrow \Omega'\, s$ implies $n > 0$ and $\Omega' \approx_{\mathrm{I}} n-1$.*

Proof. □

THEOREM 5.53 (Small-step adequacy).
Preservation *If $\Omega \approx_{\mathrm{D}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \approx_{\mathrm{D}} n$.*
Progress *If $\Omega \approx_{\mathrm{D}} n$, then either:*

- *$\Omega \longrightarrow \Omega'$;*
- *$n = 0$ and $\Omega = z$; or*
- *$n = n' + 1$ and $\Omega = \Omega'\, s$ for some n' and Ω' such that $\Omega' \approx_{\mathrm{I}} n'$.*

5.20

COROLLARY 5.54 (Big-step adequacy of decrements). *If $\Omega \approx_{\mathrm{D}} n$, then:*

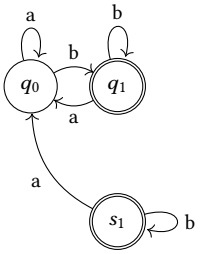
- *$\Omega \Longrightarrow \underrightarrow{z}$ if, and only if, $n = 0$;*
- *$\Omega \Longrightarrow \Omega'\, \underrightarrow{s}$ for some Ω' such that $\Omega' \approx_{\mathrm{I}} n-1$, if $n > 0$; and*
- *$\Omega \Longrightarrow \Omega'\, \underrightarrow{s}$ only if $n > 0$ and $\Omega' \approx_{\mathrm{I}} n-1$.*

5.21

5.21.1 *Automata*

1. Traces do not imply DFA transitions:

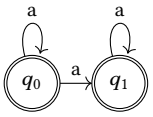
$$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \mathbin{\&} (b \setminus \hat{q}_1) \mathbin{\&} (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_0) \mathbin{\&} (b \setminus \hat{q}_1) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_1 \triangleq (a \setminus \hat{q}_0) \mathbin{\&} (b \setminus \hat{s}_1) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$



Notice that $b\,\hat{q}_0 \Longrightarrow \hat{q}_1 = \hat{s}_1$ but s_1 is not reachable from q_0. ($\hat{q}_1 = \hat{s}_1$ is proved coinductively.)

2. NFA bisimilarity does not imply equality of encodings:

$$\hat{q}_0 \triangleq (a \setminus (\hat{q}_0 \mathbin{\&} \hat{q}_1)) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_1) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$



Notice that q_0 and q_1 are bisimilar, as witnessed by the reflexive closure of $\{(q_0, q_1)\}$. However, $\hat{q}_0 \neq \hat{q}_1$.

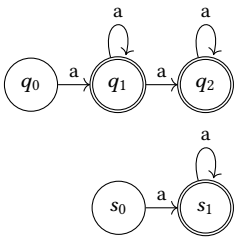
3. NFA similarity does not imply reduction. In the above example, NFA states q_0 and q_1 are bisimilar, andhence q_1 simulates q_0 (and vice versa). However, neither $\hat{q}_0 \Longrightarrow \hat{q}_1$ nor $\hat{q}_1 \Longrightarrow \hat{q}_0$ hold.

4. Even if an alternative, flatter encoding is used, NFA similarity does not imply reduction. Consider the following NFAs:

$$\hat{q}_0 \triangleq (a \setminus \hat{q}_1) \mathbin{\&} (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_1) \mathbin{\&} (a \setminus \hat{q}_2) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$
$$\hat{q}_2 \triangleq (a \setminus \hat{q}_2) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$

and

$$\hat{s}_0 \triangleq (a \setminus \hat{s}_1) \mathbin{\&} (\epsilon \setminus \top)$$
$$\hat{s}_1 \triangleq (a \setminus \hat{s}_1) \mathbin{\&} (\epsilon \setminus \mathbf{1})$$



As witnessed by the relation $\{(q_0, s_0), (q_1, s_1), (q_2, s_1)\}$, state s_0 simulates q_0. However, $\hat{q}_0 \not\Longrightarrow \hat{s}_0$. Essentially, similarity and reduction do not coincide because similarity is successor-congruent, whereas reduction is not $\setminus$-congruent.

5. Focusing with eager inversion does not solve this problem. For DFAs, we would be able to prove:

   - q and s are bisimular if, and only if, $\hat{q} = \hat{s}$.
   - $q \sim^{-1} \xrightarrow{a} \sim q'$ if, and only if, $a\,\hat{q} \longrightarrow \hat{q}'$.

6. For NFAs, we will be able to prove:

   - q and s are bisimular if, and only if, $\hat{q} \cong \hat{s}$.
   - $q \sim^{-1} \xrightarrow{a} \sim q'$ if, and only if, $a\,\hat{q} \cong^{-1} \longrightarrow \cong \hat{q}'$.

5.21.2 *Extended example: NFAs*

As an example of ordered rewriting, consider a specification of NFAs. Recall from chapter 2 the NFA (repeated in the adjacent figure) that accepts exactly those words, over the alphabet $\Sigma = \{a, b\}$, that end with b. We may represent that NFA as a rewriting specification using a collection of recursive definitions, one for each of the NFA's states:[72]

$$\hat{q}_0 \triangleq (a \setminus \hat{q}_0) \,\&\, (b \setminus (\hat{q}_0 \,\&\, \hat{q}_1)) \,\&\, (\epsilon \setminus \top)$$
$$\hat{q}_1 \triangleq (a \setminus \hat{q}_2) \,\&\, (b \setminus \hat{q}_2) \,\&\, (\epsilon \setminus \mathbf{1})$$
$$\hat{q}_2 \triangleq (a \setminus \hat{q}_2) \,\&\, (b \setminus \hat{q}_2) \,\&\, (\epsilon \setminus \top)$$

The NFA's acceptance of words is represented by the existence of traces. For example, because the word ab ends with b, a trace $\epsilon\, b\, a\, \hat{q}_0 \Longrightarrow \cdot$ exists. On the other hand, $\epsilon\, a\, b\, \hat{q}_0 \not\Longrightarrow \cdot$ because the word ba does not end with b.

More generally, an NFA $\mathcal{A} = (Q, \longrightarrow, F)$ over an input alphabet Σ can be represented as the ordered rewriting specification in which each state $q \in Q$ corresponds to a recursively defined proposition $\hat{q}$:

$$\hat{q} \triangleq \left(\mathop{\&}_{a \in \Sigma} \left(a \setminus \&_{q'_a \in \Delta(q,a)}\, \hat{q}'_a \right) \right) \,\&\, \left(\epsilon \setminus \hat{F}(q) \right) \quad \text{where} \quad \hat{F}(q) = \begin{cases} \mathbf{1} & \text{if } q \in F \\ \top & \text{if } q \notin F\,. \end{cases}$$

After defining a representation, $\underline{w}$, of words w (see adjacent figure), we may state and prove that ordered rewriting under these definitions is sound and complete with respect to the NFA semantics given in chapter 2.

THEOREM 5.55. • *$q \xrightarrow{a} q'$ if, and only if, $a\, \hat{q} \Longrightarrow \hat{q}'$.*

• *$q \in F$ if, and only if, $\epsilon\, \hat{q} \Longrightarrow \cdot$.*

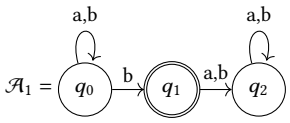


Figure 5.35: An NFA that accepts, from state q_0, exactly those words that end with b. (Repeated from fig. 2.1.)

[72] Should I include $\,\&\, (\epsilon \setminus \top)$?

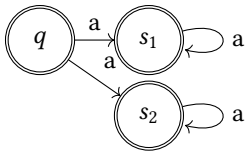
$$\underline{\epsilon} = \cdot$$
$$\underline{a\, w} = \underline{w}\, a$$

Figure 5.36: Words as ordered contexts

FALSE CLAIM 5.56. *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be an NFA over the input alphabet Σ. Then:*

- $q \xrightarrow{a}\sim s'$ *if, and only if,* $a\,\hat{q} \Longrightarrow \hat{s}'$.
- $q \sim s$ *if, and only if,* $\hat{q} = \hat{s}$.

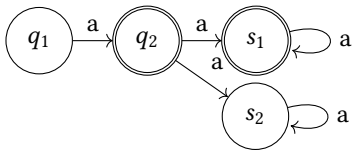
Counterexample. First, $q \sim s$ does not imply $\hat{q} = \hat{s}$. Consider the following NFA and its corresponding definitions:



$$\hat{q} \triangleq (a \setminus \hat{s}_1)\ \&\ (a \setminus \hat{s}_2)\ \&\ (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_1 \triangleq (a \setminus \hat{s}_1)\ \&\ (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_2 \triangleq (a \setminus \hat{s}_2)\ \&\ (\epsilon \setminus \mathbf{1})$$

Observe that the universal binary relation on states is a bisimulation: every state has an a-successor and every state is an accepting state. Therefore, all pairs of states are bisimilar; in particular, $q \sim s_1$. However, $\hat{q} \neq \hat{s}_1$.

Second, $a\,\hat{q} \Longrightarrow \hat{s}'$ does not imply $q \xrightarrow{a}\sim s'$. Consider the following NFA and its corresponding definitions:



$$\hat{q}_1 \triangleq (a \setminus \hat{q}_2)\ \&\ (\epsilon \setminus \top)$$
$$\hat{q}_2 \triangleq (a \setminus \hat{s}_1)\ \&\ (a \setminus \hat{s}_2)\ \&\ (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_1 \triangleq (a \setminus \hat{s}_1)\ \&\ (\epsilon \setminus \mathbf{1})$$
$$\hat{s}_2 \triangleq (a \setminus \hat{s}_2)\ \&\ (\epsilon \setminus \top)$$

Observe that $a\,\hat{q}_1 \Longrightarrow \hat{q}_2 \Longrightarrow \hat{s}_1$. However, $q_2 \not\sim s_1$, and so $q_1 \xrightarrow{a}\sim s_1$ does *not* hold. To see why $q_2 \not\sim s_1$, notice that $q_2 \xrightarrow{a} s_2 \notin F$ is not matched from s_1, which has only $s_1 \xrightarrow{a} s_1 \in F$. □

DEFINITION 5.1. A binary relation $\mathcal{R}$ on states is a simulation if:

- $s\ \mathcal{R}^{-1} \xrightarrow{a} q'$ implies $s \xrightarrow{a} \mathcal{R}^{-1}\ q'$; and
- $s\ \mathcal{R}^{-1}\ q \in F$ implies $s \in F$.

Similarity, $\lesssim$, is the largest simulation.

LEMMA 5.57. *If $\hat{q} \Longleftarrow \hat{s}$, then $q \lesssim s$.*

Proof. We must check two properties:

- Suppose that $\hat{s} \Longrightarrow \hat{q}$ and $q \xrightarrow{a} q'_a$ for some state q'_a; we must show that $s \xrightarrow{a} s'_a$ and $\hat{s}'_a \Longleftarrow \hat{q}'_a$, for some state s'_a. According to the definition, the definiens of $\hat{q}$ contains a clause $(a \setminus \hat{q}'_a)$. Because $\hat{s} \Longrightarrow \hat{q}$, the definiens of $\hat{s}$ also contains the clause $(a \setminus \hat{q}'_a)$. It follows that $s \xrightarrow{a} q'_a$ and $\hat{q}'_a \Longleftarrow \hat{q}'_a$.
- Suppose that $\hat{s} \Longrightarrow \hat{q}$ and $q \in F$; we must show that $s \in F$. According to the definition, the definiens of $\hat{q}$ contains a clause $(\epsilon \setminus \mathbf{1})$. Because $\hat{s} \Longrightarrow \hat{q}$, the definiens of $\hat{s}$ also contains the clause $(\epsilon \setminus \mathbf{1})$. It follows that $s \in F$. □

THEOREM 5.58 (Adequacy). *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be an NFA over the input alphabet Σ. If $q \xrightarrow{a} q'$, then $a\,\hat{q} \Longrightarrow \hat{q}'$. Moreover, if $a\,\hat{q} \Longrightarrow \hat{s}'$, then $q \xrightarrow{a}\gtrsim s'$.*

Proof. The first part follows by construction.

To prove the second part, suppose $a\,\hat{q} \Longrightarrow \hat{s}'$. By the lemma, $\hat{q} \Longrightarrow (a \setminus B)\,\Omega'_a$ and $B\,\Omega'_a \Longrightarrow \hat{s}'$ for some B and Ω'_a. By inversion, $\Omega'_a = \cdot$ and $B = \hat{q}'_a$ for some state q'_a such that $q \xrightarrow{a} q'_a$. Therefore, $\hat{q}'_a \Longrightarrow \hat{s}'$. By the lemma, $s' \lesssim q'_a$ and so $q \xrightarrow{a}\gtrsim s'$. □

THEOREM 5.59 (Adequacy). *Let $\mathcal{A} = (Q, \longrightarrow, F)$ be an NFA over the input alphabet Σ. Then:*

1. *If $q \xrightarrow{a}\sim s'$, then $a\,\hat{q} \Longrightarrow \hat{s}'$.*
2. *If $q \sim s$, then $\hat{q} = \hat{s}$.*
3. *If $\hat{q} = \hat{s}$, then $q \sim s$.*
4. *If $a\,\hat{q} \Longrightarrow \hat{s}'$, then $q \xrightarrow{a}\sim s'$.*

Proof. 1. Suppose that $q \xrightarrow{a} q' \sim s'$; we must show that $a\,\hat{q} \Longrightarrow \hat{s}'$. By construction, $a\,\hat{q} \Longrightarrow \hat{q}'$. It follows from part [...] that $\hat{q}' = \hat{s}'$, and so $a\,\hat{q} \Longrightarrow \hat{s}'$.

2. Suppose $q \sim s$; we must show that $\hat{q} = \hat{s}$.

   - Choose an arbitrary symbol $a \in \Sigma$. If $q \xrightarrow{a} q'_a$, then there exists an NFA state s'_a such that $s \xrightarrow{a} s'_a \sim^{-1} q'_a$, and, by the coinductive hypothesis, $\hat{q}'_a = \hat{s}'_a$. Conversely, if $s \xrightarrow{a} s'_a$, then there exists an NFA state q'_a such that $q \xrightarrow{a} q'_a$ and $\hat{q}'_a = \hat{s}'_a$.
   - Also, q is an accepting state if and only if s is an accepting state.

   Therefore, the definientia of $\hat{q}$ and $\hat{s}$ are equal, and, by the equirecursive interpretation of definitions, so are the definienda $\hat{q}$ and $\hat{s}$.

3. Suppose that $\hat{s} = \hat{q}$ and $q \xrightarrow{a} q'$; we must show that $s \xrightarrow{a} s'$ and $\hat{s}' = \hat{q}'$, for some NFA state s'. By its definition, the definiens of $\hat{q}$ therefore contains the clause $(a \setminus \hat{q}')$. Because $\hat{s} = \hat{q}$, the definiens of $\hat{s}$ must also contain a clause $(a \setminus \hat{s}')$ for some state s' such that $s \xrightarrow{a} s'$ and $\hat{s}' = \hat{q}'$.

   Symmetrically, if $\hat{q} = \hat{s}$ and $s \xrightarrow{a} s'$, then $q \xrightarrow{a} q'$ and $\hat{q}' = \hat{s}'$, for some state q'.

4. Suppose $a\,\hat{q} \Longrightarrow \hat{q}'$. By the lemma, $a\,\hat{q} \Longrightarrow (a \setminus B)\,\Omega'_a$ and $B\,\Omega'_a \Longrightarrow \hat{q}'$ for some B and Ω'_a. By inversion, $B = \hat{q}'_a$ and $\Omega'_a = \cdot$. Therefore, $\hat{q}'_a \Longrightarrow \hat{q}'$. How to show that $q'_a \sim q'$?

□

Proof. By coinduction on $q \sim s$.

- Suppose $\hat{s} = \hat{q}$ and $q \xrightarrow{a} q'$; we must show that $s \xrightarrow{a} s'$ and $\hat{s}' = \hat{q}'$ for some NFA state s'. It follows from the coinductive hypothesis that $a\,\hat{s} = a\,\hat{q} \Longrightarrow \hat{q}'$.

□

Proof. In the left-to-right directions, by unrolling the definition of $\hat{q}$ (and a structural induction on the word w).

In the right-to-left directions, by structural induction on the given trace, using the following lemma:

> If $a\,\Omega \Longrightarrow \Omega''$ and there is no Ω_0'' for which $\Omega'' = a\,\Omega_0''$, then $\Omega \Longrightarrow (a \setminus B)\,\Omega'$ for some B and Ω' such that $B\,\Omega' \Longrightarrow \Omega''$.

Assume that $a\,\hat{q} \Longrightarrow \hat{q}'$. Using the above lemma, $\hat{q} \Longrightarrow (a \setminus B)\,\Omega'$ for some B and Ω' such that $B\,\Omega' \Longrightarrow \hat{q}'$. By inversion on the trace from $\hat{q}$, it must be that $B = \&_{q'_a \in \Delta(q,a)}\,\hat{q}'_a$ and $\Omega' = \cdot$. Further inversion on the trace from $B\,\Omega'$ establishes that $q' \in \Delta(q, a)$ and hence $q \xrightarrow{a} q'$. □

$$\hat{q} \triangleq \left(\mathop{\&}_{a \in \Sigma} \left(a \setminus \hat{q}'_a \bullet \hat{v}_a\right)\right) \& \left(\epsilon \setminus \hat{\rho}(q)\right) \quad \text{where} \quad q'_a = \delta(q, a) \text{ and } v_a = \sigma(q, a) \text{ and } v = \rho(q)$$

5.21.3 *Extended example: Binary representation of natural numbers*

As a second example, consider a rewriting specification of the binary representation of natural numbers with increment and decrement operations.

For this specification, a natural number is represented in binary by an ordered context consisting of a big-endian sequence of atoms b_0 and b_1, prefixed by the atom e; leading b_0s are permitted. For example, both $\Omega = e\,b_1$ and $\Omega = e\,b_0\,b_1$ are valid binary representations of the natural number 1.

More generally, let $\mathbb{V}(-)$ be the partial function from ordered contexts to natural numbers defined as follows; we say that the ordered context Ω *represents* natural number n if $\mathbb{V}(\Omega) = n$.

$$\begin{aligned} \mathbb{V}(e) &= 0 \\ \mathbb{V}(\Omega\,b_0) &= 2\mathbb{V}(\Omega) \\ \mathbb{V}(\Omega\,b_1) &= 2\mathbb{V}(\Omega) + 1 \end{aligned}$$

The partial function $\mathbb{V}(-)$ defines an adequate representation because, up to leading b_0s, the natural numbers and valid binary representations (*i.e.*, the domain of definition of $\mathbb{V}(-)$) are in bijective correspondence.

THEOREM 5.60 (Representational adequacy). *For all natural numbers $n \in \mathbb{N}$, there exists a context Ω such that $\mathbb{V}(\Omega) = n$. Moreover, if $\mathbb{V}(\Omega_1) = n$ and $\mathbb{V}(\Omega_2) = n$, then Ω_1 and Ω_2 are identical up to leading b_0s.*

Proof. The first part follows by induction on the natural number n; the second part follows by induction on the structure of the contexts Ω_1 and Ω_2. □

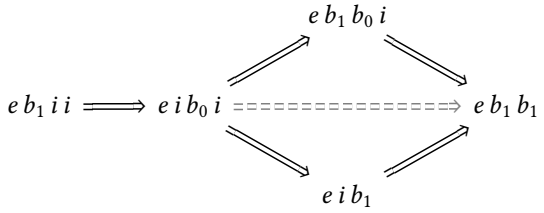
Next, we may describe an increment operation on these binary representations as an ordered rewriting specification; because of these increments, [...]. To indicate that an increment should be performed, a new, uninterpreted atom i is introduced. The previously uninterpreted atoms e, b_0, and b_1 are now given mutually recursive definitions that describe their interactions with i.

$e \triangleq e \bullet b_1 \mathbin{/} i$ To increment the counter e, introduce b_1 as a new most significant bit, resulting in the counter $e\,b_1$. That is, $e\,i \Longrightarrow e\,b_1$. Having started at value 0 (*i.e.*, $\mathbb{V}(e) = 0$), an increment results in value 1 (*i.e.*, $\mathbb{V}(e\,b_1) = 1$).

$b_0 \triangleq b_1 \mathbin{/} i$ To increment a counter that ends with least significant bit b_0, simply flip that bit to b_1. That is, $\Omega\,b_0\,i \Longrightarrow \Omega\,b_1$. Having started at value $2n$ (*i.e.*, $\mathbb{V}(\Omega\,b_0) = 2\mathbb{V}(\Omega)$), an increment results in value $2n + 1$ (*i.e.*, $\mathbb{V}(\Omega\,b_1) = 2\mathbb{V}(\Omega) + 1$).

$b_1 \triangleq i \bullet b_0 \mathbin{/} i$ To increment a counter that ends with least significant bit b_1, flip that bit to b_0 and propagate the increment on to the more significant bits as a carry. That is, $\Omega\,b_1\,i \Longrightarrow \Omega\,i\,b_0$. Having started at value $2n+1$ (*i.e.*, $\mathbb{V}(\Omega\,b_1) = 2\mathbb{V}(\Omega) + 1$), an increment results in value $2n + 2 = 2(n + 1)$ (*i.e.*, $\mathbb{V}(\Omega\,i\,b_0) = 2\mathbb{V}(\Omega) + 1$).

As an example, consider incrementing $e\,b_1$ twice, as captured by the state $e\,b_1\,i\,i$. First, processing of the leftmost increment begins: the least significant bit is flipped, and the increment is carried over to the more significant bits. This corresponds to the reduction $e\,b_1\,i\,i \Longrightarrow e\,i\,b_0\,i$. Next, either of the two remaining increments may be processed – that is, either $e\,i\,b_0\,i \Longrightarrow e\,b_1\,b_0\,i$ or $e\,i\,b_0\,i \Longrightarrow e\,i\,b_1$.



$$
\begin{array}{lll}
e\,b_1\,i\,i & e\,b_1\,i\,i & \\
\Longrightarrow e\,i\,b_0\,i & \longrightarrow e\,(i \bullet b_0 \mathbin{/} i)\,i\,i \longrightarrow e\,(i \bullet b_0)\,i \longrightarrow e\,i\,b_0\,i & \\
\Longrightarrow e\,b_1\,b_0\,i & \longrightarrow (e \bullet b_1 \mathbin{/} i)\,i\,b_0\,i \longrightarrow (e \bullet b_1)\,b_0\,i \longrightarrow e\,b_1\,b_0\,i & \\
\Longrightarrow e\,b_1\,b_1 & \longrightarrow e\,b_1\,(b_1 \mathbin{/} i)\,i \longrightarrow e\,b_1\,b_1 &
\end{array}
$$

$$\frac{}{e \approx_{\mathrm{V}} 0} \qquad \frac{\Omega \approx_{\mathrm{V}} n}{\Omega\, b_0 \approx_{\mathrm{V}} 2n} \qquad \frac{\Omega \approx_{\mathrm{V}} n}{\Omega\, b_1 \approx_{\mathrm{V}} 2n+1}$$

THEOREM 5.61 (Adequacy). *If $\Omega \approx_{\mathrm{V}} n$ and $\Omega\, i \Longrightarrow \Omega'$, then $\Omega' \Longrightarrow\!\approx_{\mathrm{V}} n+1$.*

Proof.
- Suppose that $e\, i \Longrightarrow \Omega'$; we must show that $\Omega' \Longrightarrow\!\approx_{\mathrm{V}} 1$.
- Suppose that $\Omega\, b_0\, i \Longrightarrow \Omega'$ and $\Omega \approx_{\mathrm{V}} n$; we must show that $\Omega' \Longrightarrow\!\approx_{\mathrm{V}} 2n$.

□

$$\frac{\Omega \approx_{\mathrm{V}} n}{\Omega \approx_{\mathrm{I}} n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, i \approx_{\mathrm{I}} n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_0 \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_1 \approx_{\mathrm{I}} 2n+1}$$

$$\frac{\Omega_L\, \alpha\, \Omega_R \approx_{\mathrm{I}} n \quad (\alpha \triangleq A) \in \Sigma}{\Omega_L\, A\, \Omega_R \approx_{\mathrm{I}} n}$$

THEOREM 5.62 (Preservation). *If $\Omega \approx_{\mathrm{I}} n$ and $\Omega \longrightarrow \Omega'$, then $\Omega' \Longrightarrow\!\approx_{\mathrm{I}} n$.*

Proof.
- Suppose that $\Omega_0 \approx_{\mathrm{I}} n$ and $\Omega = \Omega_0\, i \longrightarrow \Omega'$; we must show that $\Omega' \Longrightarrow\!\approx_{\mathrm{I}} n+1$.
  - Consider the case in which $\Omega_0 \longrightarrow \Omega_0'$ and $\Omega' = \Omega_0'\, i$. By the inductive hypothesis, $\Omega_0' \Longrightarrow\!\approx_{\mathrm{I}} n$. From the increment rule, it follows that $\Omega' = \Omega_0'\, i \Longrightarrow\!\approx_{\mathrm{I}} n+1$.
  - Consider the case in which $\Omega_0 = \Omega_L\, (A_0 \,/\, i)$ and $\Omega_L\, \alpha \approx_{\mathrm{I}} n$ and $\Omega' = \Omega_L\, A_0$ such that $(\alpha \triangleq A_0 \,/\, i) \in \Sigma$. There are three subcases:
    - Consider the subcase in which $\alpha = b_0$ and $n = 2n_0$ and $\Omega_L \approx_{\mathrm{I}} n_0$. By inversion on the signature, $A_0 = b_1$. It follows that $\Omega' = \Omega_L\, b_1 \approx_{\mathrm{I}} 2n_0 + 1 = n+1$.
    - Consider the subcase in which $\alpha = b_1$ and $n = 2n_0 + 1$ and $\Omega_L \approx_{\mathrm{I}} n_0$. By inversion on the signature, $A_0 = i \bullet b_0$. It follows that $\Omega' = \Omega_L\, (i \bullet b_0) \longrightarrow \Omega_L\, i\, b_0 \approx_{\mathrm{I}} 2(n_0+1) = n+1$.
    - Consider the subcase in which $\alpha = e$ and $n = 0$ and $\Omega_L = \cdot$. By inversion on the signature, $A_0 = e \bullet b_1$. It follows that $\Omega' = e \bullet b_1 \longrightarrow e\, b_1 \approx_{\mathrm{I}} 1 = n+1$.

□

THEOREM 5.63 (Progress). *If $\Omega \approx_{\mathrm{I}} n$, then either: $\Omega \longrightarrow \Omega'$ for some Ω'; or $\Omega \approx_{\mathrm{V}} n$.*

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

$$\frac{\Omega_L\, \alpha\, \Omega_R \approx_{\mathrm{D}} n \quad (\alpha \triangleq A) \in \Sigma}{\Omega_L\, A\, \Omega_R \approx_{\mathrm{D}} n}$$

$\Omega' \approx_{\mathrm{D}} n$ if, and only if, $\Omega\, d \Longrightarrow \Omega'$ for some Ω such that $\Omega \approx_{\mathrm{I}} n$.

THEOREM 5.64 (Preservation). *If* $\Omega \approx_{\mathrm{D}} n$ *and* $\Omega \Longrightarrow \Omega'$, *then* $\Omega' \approx_{\mathrm{D}} n$.

THEOREM 5.65 (Progress). *If* $\Omega \approx_{\mathrm{D}} n$, *then either:*

- $\Omega \longrightarrow \Omega'$ *for some* Ω';
- $\Omega = \Omega'\, s$ *and* $n = n' + 1$ *and* $\Omega' \approx_{\mathrm{I}} n'$ *for some* Ω' *and* n'; *or*
- $\Omega = z$ *and* $n = 0$.

5.21.4 *Examples*

- Alternative choreography – how are these related?

$$
\begin{aligned}
p &\triangleq (i \bullet p \,/\, \underleftarrow{i}) \,\&\, (d \bullet p' \,/\, \underleftarrow{d}) \\
p' &\triangleq (\underrightarrow{z} \setminus \underrightarrow{z}) \,\&\, (\underrightarrow{s} \setminus p \bullet \underrightarrow{s}) \\
i &\triangleq (\underrightarrow{e} \setminus \underrightarrow{e} \bullet \underrightarrow{b_1}) \,\&\, (\underrightarrow{b_0} \setminus \underrightarrow{b_1}) \,\&\, (\underrightarrow{b_1} \setminus i \bullet \underrightarrow{b_0}) \\
d &\triangleq (\underrightarrow{e} \setminus \underrightarrow{z}) \,\&\, (\underrightarrow{b_0} \setminus d \bullet b_0') \,\&\, (\underrightarrow{b_1} \setminus \underrightarrow{b_0} \bullet \underrightarrow{s}) \\
b_0' &\triangleq (\underrightarrow{z} \setminus \underrightarrow{z}) \,\&\, (\underrightarrow{s} \setminus \underrightarrow{b_1} \bullet \underrightarrow{s})
\end{aligned}
$$

$$
\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, d \approx_{\mathrm{D}} n}
$$

If $\Omega\, \underleftarrow{i} \longrightarrow \Omega'$, then $\underrightarrow{\Omega}\, i \longrightarrow \underrightarrow{\Omega'}$.

5.22

6
Choreographies

As shown in chapter 4, string rewriting is a suitable framework for describing the behavior[1] of concurrent systems whose components form a monoidal structure. But these [string rewriting] descriptions are [...] high-level specifications, not implementations, because they give only a global characterization of the interactions between components.

String rewriting axioms $w \longrightarrow w'$ are strictly global in their phrasing, stating merely that any substring of the form w may be replaced, en masse, with w' – nothing is said about how this replacement is achieved. The string rewriting framework therefore implicitly suggests that a meta-level actor is responsible for coordinating and conducting the rewriting, with substrings and their constituent symbols as mere passive accessories[2].

In this chapter, our goal is to move toward a lower level of abstraction.

Accordingly, we refine the focused ordered rewriting framework of the previous chapter into one that can be given a *formula-as-process* interpretation in which [ordered] rewriting faithfully represents message-passing communication among processes that are arranged in a linear topology. Then, in **??**, we describe how a string rewriting specification may be transformed into an ordered rewriting *choreography*.

In this chapter, we refine the focused ordered rewriting framework of the previous chapter into one that can be given a *formula-as-process* interpretation in which rewriting faithfully represents message-passing communication among

As argued

Then we show how, given a mapping of symbols to either [...], choreographies can be generated from a string rewriting specification.

[1] ?

[2] word choice?

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

Negative propositions, A^-, are interpreted as message-passing processes, with positive atoms, a^+, as messages passed between them. Ordered contexts, Ω, are then configurations of processes and messages arranged in a linear topology. Finally, the positive propostions, A^+, reify ordered contexts, and so they

can be interpreted as process expressions that reify process configurations.

The ordered implications $A^+ \setminus B^-$ and B^- / A^+ are restricted to $a^+ \setminus B^-$ and B^- / a^+, respectively, so that they may cleanly be interpreted as processes that input a message a^+ from the left and right, respectively.

Each positive atom a^+ is assigned a direction, either $\underset{\leftarrow}{a}$ or $\underset{\rightarrow}{a}$, that indicates

$\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{a}$; and $A^+ \setminus B^-$ restricted to $\underset{\rightarrow}{a} \setminus B^-$ and similarly for right-handed implication.

6.1.1 *Formula-as-process*

Discuss here?

Example of $\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \mathbin{\&} \uparrow\mathbf{1}$, without explicitly relating it to the specification.

6.1.2 *Focused ordered rewriting, revisited*

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

$$\frac{\dfrac{}{[\underset{\rightarrow}{a}] \dashv\Vdash \underset{\rightarrow}{a}}\ \textsc{id}^{\underset{\rightarrow}{a}} \qquad \underset{\rightarrow}{\Omega_L}\ [B^-]\ \underset{\leftarrow}{\Omega_R} \Vdash C^+}{\underset{\rightarrow}{\Omega_L}\ \underset{\rightarrow}{a}\ [\underset{\rightarrow}{a} \setminus B^-]\ \underset{\leftarrow}{\Omega_R} \Vdash C^+}\ \setminus\text{L}'$$

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

6.1.3 *Input transitions*

Also discuss here?

$$\frac{\underset{\rightarrow}{\Omega_L}\ [A^-]\ \underset{\leftarrow}{\Omega_R} \Vdash C^+ \qquad [C^+] \dashv\Vdash \Omega'}{\underset{\rightarrow}{\Omega_L}\ [A^-]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'}$$

$$\frac{\underset{\rightarrow}{\Omega_L}\ \underset{\rightarrow}{a}\ [\Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega_L}\ [\underset{\rightarrow}{a}\ \Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega_L}\ [\Omega]\ \underset{\leftarrow}{a}\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega_L}\ [\Omega\ \underset{\leftarrow}{a}]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'}$$

$$\frac{[\Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'}{[\omega\ \Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \omega\ \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega_L}\ [\Omega] \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega_L}\ [\Omega\ \omega] \longrightarrow \Omega'\ \omega}$$

THEOREM 6.1. *If* $\underset{\rightarrow}{\Omega_L}\ [\Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'$, *then* $\underset{\rightarrow}{\Omega_L}\ \Omega\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'$. *Conversely, if* $\Omega \longrightarrow \Omega'$, *then* $[\Omega] \longrightarrow \Omega'$.

Proof. The two claims are proved by induction on the structure of the given input transition and reduction, respectively, after first proving an easy lemma:

- If $\underset{\rightarrow}{\Omega_L}\ [\Omega]\ \underset{\leftarrow}{\Omega_R} \longrightarrow \Omega'$, then $[\underset{\rightarrow}{\Omega_L}\ \Omega\ \underset{\leftarrow}{\Omega_R}] \longrightarrow \Omega'$. □

6.2 *Constructing a choreography from a specification*

Suppose that we are given a string rewriting specification that consists of axioms ? over the rewriting alphabet Σ. A *choreographing assignment* is an injection in which each symbol $a \in \Sigma$ is mapped to an ordered proposition: either an atomic proposition, $\underset{\leftarrow}{a}$ or $\underset{\rightarrow}{a}$, or a recursively defined proposition, $\hat{a}$.

Given a choregraphing assignment θ, we may construct a choregraphy from the string rewriting specification. Intuitively, each axiom is annotated according to θ, and then the resulting [...] are used to construct a family of recursive definitions, one for each $\hat{a}$ in the image of θ.

A choreography is an ordered rewriting specification that simulates the string rewriting specification [...].

Consider the recurring string rewriting specification with axioms

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

We must consistently annotate each symbol as either a left-directed atom, right-directed atom, or recursively defined proposition in sucha way that each axiom's premise w has the form $w_1\, a\, w_2$ with

$$\underset{\rightarrow}{a}\, \hat{b} \longrightarrow \hat{b} \qquad \text{and} \qquad \hat{b} \longrightarrow \cdot$$

Now we must solve for $\hat{b}$, determining a definition $\hat{b} \triangleq B^-$ such that these two rewriting steps are derivable. For the first step to be derivable, $\hat{b}$ should have a definition that is consistent with $\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}$, for

$$\underset{\rightarrow}{a}\, (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \longrightarrow \hat{b}$$

Consider the choreographing assignment θ that maps a to the atom $\underset{\rightarrow}{a}$ and b to the recursively defined proposition $\hat{b}$. Upon annotating the above string rewriting axioms according to θ, we arrive at the [...]

$$\underset{\rightarrow}{a}\, \hat{b} \longrightarrow \hat{b} \qquad \text{and} \qquad \hat{b} \longrightarrow \mathbf{1}\,.$$

$$\hat{b} \triangleq \underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b} \qquad \text{and} \qquad \hat{b} \triangleq \uparrow\mathbf{1}\,,$$

respectively. Combining these into a single definition that allows a nondeterministic choice between the two, we have

$$\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\downarrow\hat{b}) \mathbin{\&} \uparrow\mathbf{1}\,,$$

or $\hat{b} \triangleq (\underset{\rightarrow}{a} \setminus \uparrow\hat{b}) \mathbin{\&} \mathbf{1}$ if the minimally necessary shifts are elided.

By construction, this choreography is adequate with respect to the specification, in the sense that it can simulate each of the specification's possible steps and vice versa.

- $w \longrightarrow w'$ only if $\theta(w) \longrightarrow \theta(w')$ For example, just as the string rewriting specification admits $a\, b \longrightarrow b$, the ordered rewriting choreography admits

$$\theta(a\, b) = \underset{\rightarrow}{a}\, b = \underset{\rightarrow}{a}\, \big((\underset{\rightarrow}{a} \setminus \uparrow\downarrow b) \mathbin{\&} \mathbf{1}\big) \longrightarrow b = \theta(b)\,.$$

- $\theta(w) \longrightarrow \Omega'$ only if $w \longrightarrow \theta^{-1}(\Omega')$ For example, just as the ordered rewriting choreography admits $\theta(b) = b \longrightarrow \cdot$, the string rewriting specification admits $b \longrightarrow \epsilon = \theta^{-1}(\cdot)$.

6.2.1 *Formal description*

Judgments $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ and $\theta \vdash w_1\ [A^-]\ w_2 \rightsquigarrow B^-$. In both judgments, all terms before the $\rightsquigarrow$ are inputs; all terms after the $\rightsquigarrow$ are outputs.

$$\frac{}{\theta \vdash \cdot \rightsquigarrow \cdot}$$

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash w_1\ [\bullet\theta(w')]\ w_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\Sigma'(\hat{a}) = A^-)}{\theta \vdash \Sigma, w_1\ a\ w_2 \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq A^- \mathbin{\&} B^-}$$

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash w_1\ [\bullet\theta(w')]\ w_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\hat{a} \notin \operatorname{dom}\Sigma')}{\theta \vdash \Sigma, w_1\ a\ w_2 \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq B^-}$$

$$\frac{}{\theta \vdash \epsilon\ [A^+]\ \epsilon \rightsquigarrow \uparrow A^+}$$

$$\frac{\theta \vdash w_1\ [A^+]\ w_2 \rightsquigarrow B^- \quad (\theta(b) = \underrightarrow{b})}{\theta \vdash w_1\ b\ [A^+]\ w_2 \rightsquigarrow \underrightarrow{b} \setminus B^-} \qquad \frac{\theta \vdash w_1\ [A^+]\ w_2 \rightsquigarrow B^- \quad (\theta(b) = \underleftarrow{b})}{\theta \vdash w_1\ [A^+]\ b\ w_2 \rightsquigarrow B^- \mathbin{/} \underleftarrow{b}}$$

Lemma 6.2. *If $\theta \vdash w_1\ [C^+]\ w_2 \rightsquigarrow B^-$, then $\theta(w_1)\ [B^-]\ \theta(w_2) \Vdash C^+$.*

Proof. By induction over the structure of the given derivation.

Consider the case in which

$$\frac{}{\theta \vdash \epsilon\ [C^+]\ \epsilon \rightsquigarrow \uparrow C^+}.$$

Because $\theta(\epsilon) = \cdot$, it is immediate that $\theta(\epsilon)\ [\uparrow C^+]\ \theta(\epsilon) \Vdash C^+$.

Consider the case in which

$$\frac{\theta \vdash w_1\ [C^+]\ w_2 \rightsquigarrow B^- \quad (\theta(b) = \underrightarrow{b})}{\theta \vdash w_1\ b\ [C^+]\ w_2 \rightsquigarrow \underrightarrow{b} \setminus B^-}.$$

By the inductive hypothesis, $\theta(w_1)\ [B^-]\ \theta(w_2) \Vdash C^+$. Appending the $\setminus\text{L}'$ rule and observing that $\theta(b) = \underrightarrow{b}$, we see that $\theta(w_1\ b)\ [\underrightarrow{b} \setminus B^-]\ \theta(w_2) \Vdash C^+$, as required.

The case involving the ?? rule is analogous to the previous one. □

Lemma 6.3. *If $\Omega \longrightarrow_{\Sigma'} \Omega'$, then:*

- *$\Sigma'(\hat{a}) = A^-$ implies $\Omega \longrightarrow_{\Sigma', \hat{a} \triangleq A^- \mathbin{\&} B^-} \Omega'$, for all B^-; and*
- *$\hat{a} \notin \operatorname{dom}\Sigma'$ implies $\Omega \longrightarrow_{\Sigma', \hat{a} \triangleq B^-} \Omega'$, for all B^-.*

Lemma 6.4. *If $(w \longrightarrow w') \in \Sigma$ and $\theta \vdash \Sigma \rightsquigarrow \Sigma'$, then $\theta(w) \longrightarrow_{\Sigma'} \theta(w')$.*

Proof. By induction over the structure of the given choreographing derivation.

- Consider the case in which $w = w_1\, a\, w_2 \longrightarrow w'$ is the axiom in question and

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash w_1\, [\bullet\theta(w')]\, w_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\Sigma'(\hat{a}) = A^-)}{\theta \vdash \Sigma, w_1\, a\, w_2 \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq A^- \,\&\, B^-}$$

  It follows from **??** that $\theta(w_1)\, [B^-]\, \theta(w_2) \Vdash \bullet\theta(w')$, and hence $\theta(w_1)\, [A^- \,\&\, B^-]\, \theta(w_2) \Vdash \bullet\theta(w')$. And because $[\bullet\theta(w')] \dashv\Vdash \theta(w')$ and $\hat{a} \triangleq A^- \,\&\, B^-$, we have $\theta(w) = \theta(w_1)\, \hat{a}\, \theta(w_2) \longrightarrow \theta(w')$.

- Consider the case in which

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash v_1\, [\bullet\theta(v')]\, v_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\Sigma'(\hat{a}) = A^-)}{\theta \vdash \Sigma, v_1\, a\, v_2 \longrightarrow v' \rightsquigarrow \Sigma', \hat{a} \triangleq A^- \,\&\, B^-}$$

  and the axiom $w \longrightarrow w'$ comes from Σ. By the inductive hypothesis, $\theta(w) \longrightarrow_{\Sigma'} \theta(w')$. By **??**, $\theta(w) \longrightarrow_{\Sigma', \hat{a} \triangleq A^- \& B^-} \theta(w')$.

- Consider the case in which

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash v_1\, [\bullet\theta(v')]\, v_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\hat{a} \notin \operatorname{dom}\Sigma')}{\theta \vdash \Sigma, v_1\, a\, v_2 \longrightarrow v' \rightsquigarrow \Sigma', \hat{a} \triangleq B^-}$$

  and the axiom $w \longrightarrow w'$ comes from Σ. By the inductive hypothesis, $\theta(w) \longrightarrow_{\Sigma'} \theta(w')$. By **??**, $\theta(w) \longrightarrow_{\Sigma', \hat{a} \triangleq B^-} \theta(w')$. □

THEOREM 6.5. *If $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ and $w \longrightarrow_\Sigma w'$, then $\theta(w) \longrightarrow_{\Sigma'} \theta(w')$.*

Proof. Consider the case in which the given string rewriting step is derived by an axiom from Σ. The desired result follows directly from **??**.

Consider the case in which the given string rewriting step is derived by

$$w = \frac{w_1 \longrightarrow w_1'}{w_1\, w_2 \longrightarrow w_1'\, w_2}\ \longrightarrow_{\mathrm{C_L}} = w'$$

By the inductive hypothesis, $\theta(w_1) \longrightarrow \theta(w_1')$. It follows from ordered rewriting's $\longrightarrow_{\mathrm{C_L}}$ rule that $\theta(w) = \theta(w_1)\, \theta(w_2) \longrightarrow \theta(w_1')\, \theta(w_2) = \theta(w')$.

The case in which the given string rewriting step is derived by the $\longrightarrow_{\mathrm{C_R}}$ rule is analogous. □

LEMMA 6.6. *If $\theta \vdash w_1\, [A^+]\, w_2 \rightsquigarrow B^-$ and $\underrightarrow{\Omega}_1\, [B^-]\, \underleftarrow{\Omega}_2 \Vdash C^+$, then $\underrightarrow{\Omega}_1 = \theta(w_1)$ and $\underleftarrow{\Omega}_2 = \theta(w_2)$ and $A^+ = C^+$.*

Proof.

$$\frac{}{\theta \vdash \epsilon\, [A^+]\, \epsilon \rightsquigarrow \uparrow A^+ = B^-}$$

By inversion on the left-focus derivation, $\underrightarrow{\Omega}_1 = \cdot = \theta(\epsilon)$ and $\underleftarrow{\Omega}_2 = \cdot = \theta(\epsilon)$ and $A^+ = C^+$.

$$\frac{(\theta(b) = \underleftarrow{b}) \quad \theta \vdash w_1\, [A^+]\, w_2 \rightsquigarrow B^-}{\theta \vdash w_1\, b\, [A^+]\, w_2 \rightsquigarrow \underleftarrow{b} \setminus B^-}$$

By inversion on the left-focus derivation for $\underleftarrow{b} \setminus B^-$, there exists $\underrightarrow{\Omega}_1'$ such that $\underrightarrow{\Omega}_1 = \underrightarrow{\Omega}_1'\, \underrightarrow{b}$ and $\underrightarrow{\Omega}_1'\, [B^-]\, \underleftarrow{\Omega}_2 \Vdash C^+$. By the inductive hypothesis, $\underrightarrow{\Omega}_1' = \theta(w_1)$ and $\underleftarrow{\Omega}_2 = \theta(w_2)$ and $A^+ = C^+$. So $\underrightarrow{\Omega}_1 = \theta(w_1\, b)$. □

LEMMA 6.7. *If* $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ *and* $\overrightarrow{\Omega}_L\ [\hat{a}]\ \overleftarrow{\Omega}_R \Vdash C^+$ *with* $\hat{a} \in \operatorname{dom}\Sigma'$, *then there exists a string rewriting axiom* $(w_1\, a\, w_2 \longrightarrow w') \in \Sigma$ *such that* $\overrightarrow{\Omega}_L = \theta(w_1)$, $\overleftarrow{\Omega}_R = \theta(w_2)$, *and* $C^+ = \bullet\theta(w')$.

Proof. By induction over the structure of the given choreographing derivation.

- Consider the case in which

$$\frac{(\theta(b) = \hat{b}) \quad \theta \vdash v_1\ [\bullet\theta(v')]\ v_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\Sigma'(\hat{b}) = A^-)}{\theta \vdash \Sigma, v_1\, b\, v_2 \longrightarrow v' \rightsquigarrow \Sigma', \hat{b} \triangleq A^- \,\&\, B^-}\ .$$

  - Suppose that $a \neq b$. Then $\hat{a} \in \operatorname{dom}\Sigma'$. By the inductive hypothesis, there exists a string rewriting axiom $(w_1\, a\, w_2 \longrightarrow w') \in \Sigma$ such that $\overrightarrow{\Omega}_L = \theta(w_1)$, $\overleftarrow{\Omega}_R = \theta(w_2)$, and $C^+ = \bullet\theta(w')$. The larger signature, $\Sigma, v_1\, b\, v_2 \longrightarrow v'$ also contains the axiom $w_1\, a\, w_2 \longrightarrow w'$.
  - Suppose that $a = b$. There are two cases, according to whether the derivation of $\overrightarrow{\Omega}_L\ [\hat{a}]\ \overleftarrow{\Omega}_R \Vdash C^+$ ends with the $\&\text{L}_1$ or $\&\text{L}_2$ rule.
    - If the left-focus derivation ends with the $\&\text{L}_2$ rule, then $\overrightarrow{\Omega}_L\ [B^-]\ \overleftarrow{\Omega}_R \Vdash C^+$. By ??, $\overrightarrow{\Omega}_L = \theta(v_1)$ and $\overleftarrow{\Omega}_R = \theta(v_2)$ and $C^+ = \bullet\theta(v')$. Choose the axiom $w_1\, a\, w_2 \longrightarrow w'$ to be $v_1\, b\, v_2 \longrightarrow v'$.
    - If the left-focus derivation instead ends with the $\&\text{L}_1$ rule, then $\overrightarrow{\Omega}_L\ [A^-]\ \overleftarrow{\Omega}_R \Vdash C^+$. By the inductive hypothesis, $\overrightarrow{\Omega}_L = \theta(v_1)$, $\overleftarrow{\Omega}_R = \theta(v_2)$, and $C^+ = \bullet\theta(v')$ for some string rewriting axiom $(v_1\, a\, v_2 \longrightarrow v') \in \Sigma$.

- Consider the case in which

$$\frac{(\theta(a) = \hat{a}) \quad \theta \vdash w_1\ [\bullet\theta(w')]\ w_2 \rightsquigarrow B^- \quad \theta \vdash \Sigma \rightsquigarrow \Sigma' \quad (\hat{a} \notin \operatorname{dom}\Sigma')}{\theta \vdash \Sigma, w_1\, a\, w_2 \longrightarrow w' \rightsquigarrow \Sigma', \hat{a} \triangleq B^-}$$

  Because $\hat{a} \notin \operatorname{dom}\Sigma'$, the left-focus derivation is $\overrightarrow{\Omega}_L\ [B^-]\ \overleftarrow{\Omega}_R \Vdash C^+$. By ??, $\overrightarrow{\Omega}_L = \theta(w_1)$, $\overleftarrow{\Omega}_R = \theta(w_2)$, and $\bullet\theta(w') = C^+$. □

THEOREM 6.8. *If* $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ *and* $\theta(w) \longrightarrow_{\Sigma'} \Omega'$, *then* $\Omega' = \theta(w')$ *for some* w' *such that* $w \longrightarrow_\Sigma w'$.

Proof. By induction over the structure of the given ordered rewriting step.

Consider the case in which

$$\theta(w) = \frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\ \Omega_2 \longrightarrow \Omega_1'\ \Omega_2}\ \longrightarrow_{\text{C}_\text{L}} = \Omega'\ .$$

By inversion, $w = w_1\, w_2$ for some w_1 and w_2 such that $\Omega_1 = \theta(w_1)$ and $\Omega_2 = \theta(w_2)$. By the inductive hypothesis, there exists a string w_1' such that $w_1 \longrightarrow w_1'$ and $\Omega_1' = \theta(w_1')$. Let $w' = w_1'\, w_2$, and notice that $w = w_1\, w_2 \longrightarrow w_1'\, w_2 = w'$ and $\Omega' = \theta(w_1')\, \theta(w_2) = \theta(w')$, as required.

The case in which the given ordered rewriting step is derived by the $\longrightarrow_{\text{C}_\text{L}}$ rule is symmetric.

Consider the case in which

$$\theta(w) = \dfrac{\underset{\rightarrow}{\Omega}_1\,[A^-]\,\underset{\leftarrow}{\Omega}_2 \Vdash C^+ \quad [C^+] \dashv\!\shortmid\, \Omega'}{\underset{\rightarrow}{\Omega}_1\,A^-\,\underset{\leftarrow}{\Omega}_2 \longrightarrow \Omega'}$$

By inversion, $w = w_1\,a\,w_2$ for some w_1, a, and w_2 such that $\theta(w_1) = \underset{\rightarrow}{\Omega}_1$, $\theta(a) = \hat{a}$, $\hat{a} \triangleq A^-$, and $\theta(w_2) = \underset{\leftarrow}{\Omega}_2$. By **??**, $C^+ = \bullet\theta(w')$ for some w' such that $(w_1\,a\,w_2 \longrightarrow w') \in \Sigma$ is an axiom. Because $[\bullet\theta(w')] \dashv\!\shortmid\, \Omega'$ only if $\Omega' = \theta(w')$, it follows that $w = w_1\,a\,w_2 \longrightarrow w'$, with $\Omega' = \theta(w')$. □

6.2.2 *No choreography*

Not all string rewriting specifications admit a choreography. For example, the specification

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

cannot be given a choreography. More precisely, there is no choreographing assignment θ such that $\theta \vdash \Sigma \rightsquigarrow \Sigma'$ is derivable for some signature Σ'. For the sake of contradiction, suppose that θ were such a choreographing assignment. Then, for the specification's latter two axioms to be choreographable, both $\theta(a) = \hat{a}$ and $\theta(b) = \hat{b}$ must hold. In that case, however, the specification's first axiom cannot be choreographed properly because θ maps more than one of the axiom's symbols to recursively defined propositions.

6.3 *Encoding nondeterministic finite automata*

Recall from **??** our string rewriting specification of how an NFA processes its input. Given an NFA $\mathcal{A} = (Q, ?, F)$ over an input alphabet Σ, the NFA's operational semantics are adequately captured by the folllwing string rewriting axioms:

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

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

6.3.1 *A functional choreography*

One possible choreography for this specification treats the input symbols $a \in \Sigma$ as atomic propositions $\underset{\rightarrow}{a}$; states $q \in Q$ as recursively defined propostions $\hat{q}$;and the end-of-word marker ϵ as an atomic proposition $\underset{\rightarrow}{\epsilon}$. In other words, the NFA's input is treated as a sequence of messages, $\underset{\rightarrow}{\epsilon}\,\underset{\rightarrow}{a}_n \cdots \underset{\rightarrow}{a}_2\,\underset{\rightarrow}{a}_1$, and the NFA's states are treated as [recursive] processes.

$a \mapsto \underset{\rightarrow}{a}$ for all $a \in \Sigma$; $q \mapsto \hat{q}$ for all $q \in Q$; and $\epsilon \mapsto \underset{\rightarrow}{\epsilon}$.

Using this assignment, the choreography constructed from the specification consists of the following definition, one for each NFA state $q \in Q$:

$$\hat{q} \triangleq \mathop{\&}_{a \in \Sigma} \mathop{\&}_{q'_a} (\underset{\rightarrow}{a} \setminus \hat{q}'_a) \,\&\, (\underset{\rightarrow}{\epsilon} \setminus \hat{F}(q))\,.$$

COROLLARY 6.9. *If* $q \xrightarrow{a} q'_a$, *then* $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \hat{q}'_a$. *If* $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$, *then* $\Omega' \Longrightarrow \hat{q}'_a$ *for some* q'_a *that* a*-succeeds* q. *If* $q \xrightarrow{\vec{a}} q'_a$, *then* $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \hat{q}'_a$. *If* $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$, *then* $\Omega' \Longrightarrow \hat{q}'_a$ *for some* q'_a *that* a*-succeeds* q.

COROLLARY 6.10. *If* $q \xrightarrow{a} q'_a$, *then* $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \hat{q}'_a$. *If* $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \Omega'$, *then* $\Omega' = \hat{q}'_a$ *for some* q'_a *that* a*-succeeds* q.

THEOREM 6.11. *Let* $\mathcal{A} = (\ldots)$ *be a DFA over the input alphabet* Σ. $q \sim s$ *if, and only if,* $\hat{q} = \hat{s}$, *for all states* q *and* s.

Proof. ... □

But the same does not hold for NFAs.

Counterexample. ... □

6.3.2 *An object-oriented choreography*

$$\hat{a} \triangleq \mathop{\&}_{q \in Q} \mathop{\&}_{q'_a} (\underset{\leftarrow}{q'_a} \,/\, \underset{\leftarrow}{q})$$
$$\hat{\epsilon} \triangleq \mathop{\&}_{q \in Q} (\hat{F}(q) \,/\, \underset{\leftarrow}{q})\,.$$

COROLLARY 6.12. *If* $q \xrightarrow{a} q'_a$, *then* $\hat{a}\,\underset{\leftarrow}{q} \Longrightarrow \underset{\leftarrow}{q'_a}$. *If* $\hat{a}\,\underset{\leftarrow}{q} \longrightarrow \Omega'$, *then* $\Omega' \Longrightarrow \underset{\leftarrow}{q'_a}$ *for some* q'_a *that* a*-succeeds* q.

Define a relation on input symbols a and b such that $\hat{a} = \hat{b}$. Two symbols are then related exactly when they lead to the same successor states.

6.4 *Binary counters*

6.4.1 *An object-oriented choreography*

$$\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d})$$
$$\hat{b}_0 \triangleq (\uparrow\!\downarrow \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet \hat{b}'_0 \,/\, \underset{\leftarrow}{d})$$
$$\hat{b}_1 \triangleq (\underset{\leftarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\leftarrow}{i}) \,\&\, (\hat{b}_0 \bullet \underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{d})$$
$$\hat{b}'_0 \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})$$

Is the following result what we want?

COROLLARY 6.13. *If* $w \approx_{\mathrm{I}} n$, *then* $\theta(w) \Longrightarrow \theta(w')$ *and* $w' \approx_{\mathrm{V}} n$, *for some* w'. *If* $w \approx_{\mathrm{I}} n$ *and* $\theta(w) \Longrightarrow \Omega'$, *then* $\Omega' = \theta(w')$ *and* $w' \approx_{\mathrm{I}} n$, *for some* w'.

$$\hat{e} \triangleq (\hat{e} \bullet \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\hat{z} \,/\, \underset{\leftarrow}{d})$$
$$\hat{b}_0 \triangleq (\uparrow\!\downarrow \hat{b}_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet \underset{\leftarrow}{b'_0} \,/\, \underset{\leftarrow}{d})$$
$$\hat{b}_1 \triangleq (\underset{\leftarrow}{i} \bullet \hat{b}_0 \,/\, \underset{\leftarrow}{i}) \,\&\, (\hat{b}_0 \bullet \hat{s} \,/\, \underset{\leftarrow}{d})$$
$$\hat{z} \triangleq \uparrow\!\downarrow \hat{z} \,/\, \underset{\leftarrow}{b'_0}$$
$$\hat{s} \triangleq \hat{b}_1 \bullet \hat{s} \,/\, \underset{\leftarrow}{b'_0}$$

```
bin = &{ i: bin, d: hdun }
hdun = &{ x: +{ z: 1, s: bin } }

1 |- e : bin
bin |- b0 : bin
bin |- b1 : bin
1 |- z : hdun
bin |- s : hdun
```

$$\hat{z} \triangleq (\uparrow\downarrow\hat{z} \,/\, \underset{\leftarrow}{b}'_0) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{x})$$
$$\hat{s} \triangleq (\hat{b}_1 \bullet \hat{s} \,/\, \underset{\leftarrow}{b}'_0) \,\&\, (\underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{x})$$

6.4.2 *A functional choreography*

$$\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \hat{\imath} \bullet \underset{\rightarrow}{b}_0)$$
$$\hat{d} \triangleq (\underset{\rightarrow}{e} \setminus \uparrow\downarrow\hat{z}) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \hat{d} \bullet \underset{\leftarrow}{b}'_0) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \underset{\rightarrow}{b}_0 \bullet \hat{s})$$
$$\hat{z} \triangleq \uparrow\downarrow\hat{z} \,/\, \underset{\leftarrow}{b}'_0$$
$$\hat{s} \triangleq \underset{\rightarrow}{b}_1 \bullet \hat{s} \,/\, \underset{\leftarrow}{b}'_0$$

$$\hat{\imath} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \hat{\imath} \bullet \underset{\rightarrow}{b}_0)$$
$$\hat{d} \triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \hat{d} \bullet \hat{b}'_0) \,\&\, (\underset{\rightarrow}{b}_1 \setminus \underset{\rightarrow}{b}_0 \bullet \underset{\rightarrow}{s})$$
$$\hat{b}'_0 \triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \hat{b}_1 \bullet \underset{\rightarrow}{s})$$

7
From ordered rewriting to message-passing concurrency

The previous chapter introduced a process-as-formula view of the Lambek calculus, used to provide local, message-passing choreographies of the global, string rewriting specifications seen in chapter 4.

With the notion of process identity uninterpreted atomic propositions as messages and

This chapter explores the question of when two propositions have equivalent behavior under this process-as-formula view. In keeping with the large body of work on bisimilarity for message-passing processes,[1] we develop a notion of bisimilarity for ordered propositions. This *ordered rewriting bisimilarity* treats the uninterpreted atomic propositions as the sole observables, in keeping with their interpretation as messages. Messages are observable, but processes are opaque.

[1] ??.

7.1

7.2 *Ordered rewriting bisimilarity*

ATOMS ARE OBSERVABLE An ordered context Ω may be composed when surrounded by ordered contexts Ω_L and Ω_R.

Thus, in $\Omega_L\ \Omega\ \Omega_R$, we view Ω as existing within the environment formed by its surrounding contexts, Ω_L and Ω_R. The context Ω then interacts with that environment along two interfaces: the left end of Ω may interact with the right end of Ω_L, and, symmetrically, the right end of Ω may interact with the left end of Ω_R.

An atom's location and direction are crucial to its observability. For an atom to be observable, it must be possible for an external observer to receive that atom as a message. In $\underset{\leftarrow}{a}\,\Omega$ and $\Omega\,\underset{\rightarrow}{b}$, the atoms $\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{b}$, respectively, are observable, because $\Omega_O\ (A^+ \,/\, \underset{\leftarrow}{a})\ \underset{\leftarrow}{a}\,\Omega \longrightarrow \Omega_O\ A^+\ \Omega$

But those same atoms are not observable in $\Omega\,\underset{\leftarrow}{a}$ and $\underset{\rightarrow}{b}\,\Omega$.

THEOREM 7.1. *If* $\Omega = \underset{\leftarrow}{a}\,\Omega_0 \longrightarrow \Omega'$, *then* $\Omega' = \underset{\leftarrow}{a}\,\Omega_0'$ *for some* Ω_0' *such that* $\Omega_0 \longrightarrow \Omega_0'$. *Symmetrically, if* $\Omega = \Omega_0\,\underset{\rightarrow}{a} \longrightarrow \Omega'$, *then* $\Omega' = \Omega_0'\,\underset{\rightarrow}{a}$ *for some* Ω_0' *such that* $\Omega_0 \longrightarrow \Omega_0'$.

Proof. By inversion on the given reduction, making use of the fact that $\underset{\leftarrow}{a} \setminus B^-$ and $B^- \,/\, \underset{\rightarrow}{a}$ are not well-formed propositions. □

ORDERED REWRITING IS ASYNCHRONOUS Notice that $\Omega_L\,(\uparrow A^+ \,/\, \underset{\leftarrow}{a})\,(\underset{\leftarrow}{a} \bullet B^+)\,\Omega_R \Longrightarrow \Omega_L\,A^+\,B^+\,\Omega_R$ in two steps, first decomposing $\underset{\leftarrow}{a} \bullet B^+$ into $\underset{\leftarrow}{a}$ and B^+, and then using that $\underset{\leftarrow}{a}$ to decompose $\uparrow A^+ \,/\, \underset{\leftarrow}{a}$ into A^+. But the rewriting cannot occur in a single, synchronous step:

$$\Omega_L\,(\uparrow A^+ \,/\, \underset{\leftarrow}{a})\,(\underset{\leftarrow}{a} \bullet B^+)\,\Omega_R \not\longrightarrow \Omega_L\,A^+\,B^+\,\Omega_R\,.$$

For this reason, ordered rewriting is asynchronous, and we should expect the notion of bisimilarity that we develop to be similar to bisimilarity developed for the asynchronous π-calculus.[2]

[2] **Amadio+:TCS98**.

7.2.1

Because outgoing atoms are observable at a context's edges, there is a built-in notion of (immediate) output transition: a context Ω outputs $\underset{\leftarrow}{a}$ to its left exactly when $\Omega = \underset{\leftarrow}{a}\,\Omega'$, for some Ω'. Symmetrically, a context Ω outputs $\underset{\rightarrow}{b}$ to its right exactly when $\Omega = \Omega'\,\underset{\rightarrow}{b}$. We could adopt a process-calculus–like labeled transition notation for these output transitions – such as $\Omega = \underset{\leftarrow}{a}\,\Omega' \xrightarrow{\overset{a}{\leftarrow}} \Omega'$ and $\Omega = \Omega'\,\underset{\rightarrow}{a} \xrightarrow{\overset{a}{\rightarrow}} \Omega'$ – but that

A weak output transition would then be So, in this setting, Ω would have a weak output transition to Ω' if there exists a context Ω_0 such that $\Omega \Longrightarrow \underset{\leftarrow}{a}\,\Omega_0$ and $\Omega_0 \Longrightarrow \Omega'$ – or, more simply, if $\Omega \Longrightarrow \underset{\leftarrow}{a}\,\Omega'$.

7.3

This chapter marks a change in our perspective on ordered rewriting. In the previous chapter, we viewed ordered rewriting as an abstract framework for global specifications of concurrent systems, in the vein of previous work on [...]. The emphasis was placed squarely on state transformation [...].

Although useful for reasoning about abstract properties of concurrent systems, these global specifications do not immediately suggest [...]. Therefore, in this [...], we instead refine ordered rewriting into a framework for message-passing concurrency among processes with independent threads of control.

This message-passing view is obtained through a *process-as-formula*[3] reading of ordered propositions and contexts. The logical connectives are reinterpreted as process constructors, so that propositions are seen as processes; positive atomic propositions, as messages; and contexts, as process configurations.

[3] ??.

In this chapter, we would instead like to decrease the level of abstraction and view ordered rewriting as a framework for message-passing concurrency.

– specifically, message-passing among processes arranged in a chain[4] topology. With their independent threads of control, processes bring a more local

[4] linear?

character to ordered rewriting, bringing it closer to a process calculus such as the π-calculus.

This message-passing view is obtained through a *process-as-formula*[5] view of ordered propositions and contexts. The logical connectives are reinterpreted as process constructors, so that propositions are seen as processes; positive atomic propositions, as messages; and contexts, as process configurations.

[5] ??.

This chapter marks a change in our perspective on ordered rewriting. In the previous chapter, we viewed ordered rewriting as an abstract framework for concurrent state transformation, in the vein of previous work on multiset rewriting[6] [or even Petri nets[7]]. With the emphasis on transformation of the entire state, our view of concurrent computation was inherently global.

[6] ??.
[7] ??.

In this chapter, we would instead like to view ordered rewriting as a framework for message-passing concurrency – specifically, message-passing among processes arranged in a chain[8] topology. With their independent threads of control, processes bring a more local character to ordered rewriting, bringing it closer to a process calculus such as the π-calculus.

[8] linear?

This message-passing view is obtained through a *process-as-formula*[9] view of ordered propositions and contexts. The logical connectives are reinterpreted as process constructors, so that propositions are seen as processes; positive atomic propositions, as messages; and contexts, as process configurations.

[9] ??.

Interestingly, this change in perspective necessitates very few formal changes to the ordered rewriting framework. The primary change is that left- and right-handed implications are restricted to positive atoms, corresponding to the common first-order restriction that input processes receive only messages.

Despite the few formal changes, the new local [, message-passing] perspective does raise a new, important question: when do two processes have equivalent behavior? In keeping with the large body of work on bisimilarity,[10] we develop a notion of bisimilarity between ordered contexts. Several examples [...]

[10] ??.

Despite requiring only very few formal changes, the shift from global to local perspective does raise an important question: when do two processes have equivalent behavior? We answer this question by developing a notion of bisimilarity for ordered contexts. In keeping with the large body of work on bisimilarity,[11] we develop a notion of bisimilarity between ordered contexts.

[11] ??.

- Assign direction to uninterpreted atoms so they act like messagesl
- Defined atoms are like processes
- Left and right implications are restricted to messages (compare with higher-order π calculus)

7.4

$$a\,\hat{q} \longrightarrow \hat{q}'_a$$

where $q \xrightarrow{a} q'_a$, for each pair $(q, a) \in Q \times \Sigma$; and

$$\epsilon\,\hat{q} \longrightarrow \begin{cases} \mathbf{1} & \text{if } q \in F \\ \top & \text{if } q \notin F \end{cases}$$

for each $q \in Q$.

As a specification of DFAs, this works well. But as an implementation, it is significantly lacking. The rewriting axioms $a\,\hat{q} \longrightarrow \hat{q}'_a$ presume that a conductor orchestrates the interactions between input symbols and DFA states, but a local[12] implementation

[12] distributed?

This specification could be choreographed in (at least) two ways. One choreography treats the input symbols a as messages that are received by the states $\hat{q}$, acting as processes.

$$\hat{q} \triangleq (\epsilon \setminus \hat{F}(q)) \mathbin{\&} \mathop{\&}_{a \in \Sigma} (\underset{\rightarrow}{a} \setminus \hat{q}'_a)$$

Because the input word is delivered like data to the state, this choreography has a functional flavor.

Another choreography of the same specification is dual, treating the input symbols as processes

$$a \triangleq \mathop{\&}_{q \in Q} (q'_a \mathbin{/} \underset{\leftarrow}{q}) \quad \text{and} \quad \epsilon \triangleq \mathop{\&}_{q \in Q} (\hat{F}(q) \mathbin{/} \underset{\leftarrow}{q})$$

7.5

To interpret polarized ordered propositions as processes, we adapt the *process-as-formula* view of logical connectives initiated by **??**. The logical connectives are read as process constructors, so that positive atomic propositions may be seen as messages; negative propositions, [may be seen] as processes; ordered contexts, [may be seen] as process configurations with a chain topology; and positive propositions, [may be seen] as processes that reify those configurations.

To keep the interpretation as simple as possible, we introduce three syntactic restrictions on the ordered propositions. Each of these restrictions may be relaxed at the expense of some additional complexity, as we will discuss in **??**.

First, each positive atom is consistently assigned a direction, either left-directed, $\underset{\leftarrow}{a}$, or right-directed, $\underset{\rightarrow}{a}$. When positive atoms are viewed as messages, these directions indicate the message's sender and intended recipient. For example, in the context $\downarrow C^-\ \underset{\leftarrow}{a}\ B^+$, the right-to-left direction of $\underset{\leftarrow}{a}$ incicates that B^+ was the sender and $\downarrow C^-$ is the intended recipient.

Second, recursively defined *positive* propositions are disallowed.[13]

[13] is this necessary?

Third, the left- and right-handed implications are restricted to accept only atoms with an incoming direction: $\underrightarrow{a} \setminus B^-$ and $B^- / \underleftarrow{a}$. [In conjunction with atoms' directions,] this acts as a mild form of typing – an input process may receive only intended messages. Something like $\underleftarrow{a}\,(a \setminus \uparrow B^+) \longrightarrow B^+$ should *not* be possible, because its process-as-formula reading

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

Concatenation of contexts, $\Omega_1\,\Omega_2$, is viewed as end-to-end composition of process configurations; the empty context, $\cdot$, is the empty process configuration; and [...].

To keep the interpretation as simple as possible, we introduce three syntactic restrictions on propositions. Each of these restrictions may be relaxed at the expense of some additional complexity, as we will discuss in **??**.

First, each positive atom is consistently assigned a direction, either left-directed, $\underleftarrow{a}$, or right-directed, $\underrightarrow{a}$. Because Second, recursively defined *positive* propositions are disallowed. Thus, the positive propositions are generated by the following grammar.

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

Atoms $\underleftarrow{a}$ and $\underrightarrow{a}$ are viewed as left- and right-directed messages; ordered conjunction, $A^+ \bullet B^+$, denotes end-to-end composition of processes A^+ and B^+; $\mathbf{1}$ denotes the terminating

$\Omega_1\,\Omega_2$	end-to-end composition of configurations
$\cdot$	empty configuration
A^+	
$\underrightarrow{a}$	right-directed message
$\underleftarrow{a}$	left-directed message
$A^+ \bullet B^+$	process composition
$\mathbf{1}$	terminating process
$\downarrow A^-$	
$\alpha^- \triangleq A^-$	recursively defined process
$\underrightarrow{a} \setminus B^-$	receive $\underrightarrow{a}$ from the left, then continue as B^-
$B^- / \underleftarrow{a}$	receive $\underleftarrow{a}$ from the right, then continue as B^-
$A^- \,\&\, B^-$	nondeterministically choose to continue as A^- or B^-
$\top$	
$\uparrow A^+$	
$\Omega_1\,\Omega_2$	composition of configurations Ω_1 and Ω_2
$\cdot$	empty configuration
A^+	single process configuration

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

Atoms' directions act as a very mild form of typing. The left- and right-handed implications are restricted to accept only atoms with an incoming

direction: $\underset{\rightarrow}{a} \setminus B^-$ and $B^- / \underset{\leftarrow}{a}$. The full syntax of negative propositions is thus:

$$\text{NEGATIVE PROPS.} \quad A^- ::= \alpha^- \mid \underset{\rightarrow}{a} \setminus B^- \mid B^- / \underset{\leftarrow}{a} \mid A^- \,\&\, B^- \mid \top \mid \uparrow A^+ ,$$

with equirecursively defined negative propositions $\alpha^- \triangleq A^-$.

acting like recursively defined processes.

In addition to fully general ordered contexts of positive propositions, it will also be useful to characterize two refinements: contexts that contain only atoms of one direction or the other. We use an arrow decoration to indicate the direction.

$$\begin{aligned} \text{ORDERED CONTEXTS} \quad & \Omega ::= \Omega_1\, \Omega_2 \mid \cdot \mid A^+ \\ \text{RIGHT-DIRECTED} \quad & \underset{\rightarrow}{\Omega} ::= \underset{\rightarrow}{\Omega}_1\, \underset{\rightarrow}{\Omega}_2 \mid \cdot \mid \underset{\rightarrow}{a} \\ \text{LEFT-DIRECTED} \quad & \underset{\leftarrow}{\Omega} ::= \underset{\leftarrow}{\Omega}_1\, \underset{\leftarrow}{\Omega}_2 \mid \cdot \mid \underset{\leftarrow}{a} \end{aligned}$$

Having restricted the premises of left- and right-handed implications to incoming atoms, $\underset{\rightarrow}{a}$ and $\underset{\leftarrow}{a}$, respectively, the left focus judgment and its rules may be refined. The judgment is now $\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+$, because [...inputs can only be incoming messages...]. Other than this refinement, the inference rules remain essentially the same as in **??**. The revised

7.6 *Input transitions*

With the above restriction of left- and right-handed implications to atomic premises of hte

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, [\downarrow A^-]\, \underset{\leftarrow}{\Omega}_R \longrightarrow C^+}$$

$$\frac{\underset{\rightarrow}{\Omega}_L\, \underset{\rightarrow}{a}\, [\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\underset{\rightarrow}{a}\, \Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [\Omega]\, \underset{\leftarrow}{a}\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\Omega\, \underset{\leftarrow}{a}]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}$$

$$\frac{[\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{[A^+\, \Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow A^+\, \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [\Omega] \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\Omega\, A^+] \longrightarrow \Omega'\, A^+}$$

In its most basic form, an input transition derives from the inputs required by [...].

The following theorem relates input transitions to reductions.

THEOREM 7.2. *If* $\underset{\rightarrow}{\Omega}_L\, [\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'$, *then* $\underset{\rightarrow}{\Omega}_L\, \Omega\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'$. *Conversely, if* $\Omega \longrightarrow \Omega'$, *then there exist* Ω_L *and* Ω_R *such that either:*

- $\Omega = \Omega_L\, \underset{\rightarrow}{\Delta}_L\, \Omega_0\, \underset{\leftarrow}{\Delta}_R\, \Omega_R$ *and* $\underset{\rightarrow}{\Delta}_L\, [\Omega_0]\, \underset{\leftarrow}{\Delta}_R \longrightarrow \Omega_0'$ *and* $\Omega' = \Omega_L\, \Omega_0'\, \Omega_R$, *for some* $\underset{\rightarrow}{\Delta}_L$, Ω_0, $\underset{\leftarrow}{\Delta}_R$, *and* Ω_0';
- $\Omega = \Omega_L\, (A^+ \bullet B^+)\, \Omega_R$ *and* $\Omega' = \Omega_L\, A^+\, B^+\, \Omega_R$, *for some* A^+ *and* B^+; *or*
- $\Omega = \Omega_L\, \mathbf{1}\, \Omega_R$ *and* $\Omega' = \Omega_L\, \Omega_R$.

Figure 7.1: A weakly focused ordered rewriting framework

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

$$\text{NEGATIVE PROPS.} \quad A^- ::= \alpha^- \mid \underset{\rightarrow}{a} \setminus B^- \mid B^- / \underset{\leftarrow}{a} \mid A^- \mathbin{\&} B^- \mid \top \mid {\uparrow}A^+$$

$$\text{ORDERED CONTEXTS} \quad \Omega ::= \Omega_1\, \Omega_2 \mid \cdot \mid A^+$$
$$\underset{\rightarrow}{\Omega} ::= \underset{\rightarrow}{\Omega}_1\, \underset{\rightarrow}{\Omega}_2 \mid \cdot \mid \underset{\rightarrow}{a}$$
$$\underset{\leftarrow}{\Omega} ::= \underset{\leftarrow}{\Omega}_1\, \underset{\leftarrow}{\Omega}_2 \mid \cdot \mid \underset{\leftarrow}{a}$$

REWRITING: $\Omega \longrightarrow \Omega'$ AND $\Omega \Longrightarrow \Omega'$

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, {\downarrow}A^-\, \underset{\leftarrow}{\Omega}_R \longrightarrow C^+}\ {\downarrow}\text{D} \qquad \frac{}{A^+ \bullet B^+ \longrightarrow A^+\, B^+}\ \bullet\text{D} \qquad \frac{}{\mathbf{1} \longrightarrow \cdot}\ \mathbf{1}\text{D}$$

(no $\oplus$D and $\mathbf{0}$D rules)

$$\frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\, \Omega_2 \longrightarrow \Omega_1'\, \Omega_2}\ \longrightarrow\!\text{C}_\text{L} \qquad \frac{\Omega_1 \longrightarrow \Omega_1'}{\Omega_1\, \Omega_2 \longrightarrow \Omega_1'\, \Omega_2}\ \longrightarrow\!\text{C}_\text{R}$$

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

LEFT FOCUS: $\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+$

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

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

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

Figure 7.2: A weakly focused ordered rewriting framework

INPUT TRANSITION: $\underset{\rightarrow}{\Omega}_L\, [\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'$

$$\frac{\underset{\rightarrow}{\Omega}_L\, [A^-]\, \underset{\leftarrow}{\Omega}_R \Vdash C^+}{\underset{\rightarrow}{\Omega}_L\, [{\downarrow}A^-]\, \underset{\leftarrow}{\Omega}_R \longrightarrow C^+} \qquad \frac{\underset{\rightarrow}{\Omega}_L\, \underset{\rightarrow}{a}\, [\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\underset{\rightarrow}{a}\, \Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [\Omega]\, \underset{\leftarrow}{a}\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\Omega\, \underset{\leftarrow}{a}]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}$$

$$\frac{[\Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow \Omega'}{[A^+\, \Omega]\, \underset{\leftarrow}{\Omega}_R \longrightarrow A^+\, \Omega'} \qquad \frac{\underset{\rightarrow}{\Omega}_L\, [\Omega] \longrightarrow \Omega'}{\underset{\rightarrow}{\Omega}_L\, [\Omega\, A^+] \longrightarrow \Omega'\, A^+}$$

Proof. By structural induction on the given input transition or reduction, respectively. □

LEMMA 7.3. *If* $\underrightarrow{\Delta}_L\ [\underrightarrow{a}\ \Omega]\ \underleftarrow{\Delta}_R \longrightarrow \Omega'$, *then either:*

- $\underrightarrow{a}$ *satisfies an input demand – i.e.,* $\underrightarrow{\Delta}_L\ \underrightarrow{a}\ [\Omega]\ \underleftarrow{\Delta}_R \longrightarrow \Omega'$; *or*
- $\underrightarrow{a}$ *does not participate in the input transition – i.e.,* $\underrightarrow{\Delta}_L = \cdot$ *and* $\Omega' = \underrightarrow{a}\ \Omega'_a$ *for some* Ω'_a *such that* $[\Omega]\ \underleftarrow{\Delta}_R \longrightarrow \Omega'_a$.

Symmetrically, if $\underrightarrow{\Delta}_L\ [\Omega\ \underleftarrow{a}]\ \underleftarrow{\Delta}_R \longrightarrow \Omega'$, *then either:*

- $\underleftarrow{a}$ *satisfies an input demand – i.e.,* $\underrightarrow{\Delta}_L\ [\Omega]\ \underleftarrow{a}\ \underleftarrow{\Delta}_R \longrightarrow \Omega'$; *or*
- $\underleftarrow{a}$ *does not participate in the input transition – i.e.,* $\underleftarrow{\Delta}_R = \cdot$ *and* $\Omega' = \Omega'_a\ \underleftarrow{a}$ *for some* Ω'_a *such that* $\underrightarrow{\Delta}_L\ [\Omega] \longrightarrow \Omega'_a$.

Proof. By structural induction on the given input transition. □

7.7 *Rewriting bisimilarity*

With the shift from a global, state transformation view of ordered rewriting to a local, "formula-as-process" view, it is now possible to consider how the individual "formula-as-process" – or, more generally, "context-as-configuration" – components behave and how they interact with each other. Because each component has its own, local thread of control, we can describe its behavior only to the extent that its behavior is observable. To the extent that its behavior can be witnessed by an external observer

Now that we can decompose concurrent systems into individual "formula-as-process" – or "context-as-configuration" – components,

Intuitively, for example, the contexts[14] $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$ and $\underrightarrow{b}$ should be behaviorally equivalent: an internal reduction transforms $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$ into $\underrightarrow{b}$, and no other interactions – reductions or input or output transitions – are possible from $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$. As another example, $\underrightarrow{a} \setminus (\underleftarrow{c} \,/\, \underrightarrow{b})$ and $(\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b}$ should also be behaviorally equivalent, intuitively because they are logically equivalent.

[14] configurations?

Following the vast literature on various forms of bisimilarity[15], we will develop a notion of *rewriting bisimilarity* on ordered contexts. First, we need a few auxiliary definitions.

[15] See ?? for a survey.

related to Deng et al.

DEFINITION 7.1 (Framed binary relations). Let $\mathcal{R}$ be a binary relation over ordered contexts. Given ordered contexts Δ_L and Δ_R, let $(\Delta_L \mathcal{R} \Delta_R)$ be the least binary relation such that:

$$\frac{\Omega\ \mathcal{R}\ \Omega'}{\Delta_L\ \Omega\ \Delta_R\ (\Delta_L \mathcal{R} \Delta_R)\ \Delta_L\ \Omega'\ \Delta_R}$$

Furthermore, let $[\mathcal{R}]$ be the input contextual closure of $\mathcal{R}$ – that is, $\Omega\ [\mathcal{R}]\ \Delta$ if and only if $\Omega\ (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R)\ \Delta$ for some $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$. Equivalently, $[\mathcal{R}]$ is the least binary relation such that:

$$\frac{\Omega\ \mathcal{R}\ \Delta}{\Omega\ [\mathcal{R}]\ \Delta} \qquad \frac{\Omega\ [\mathcal{R}]\ \Delta}{\underrightarrow{a}\ \Omega\ [\mathcal{R}]\ \underrightarrow{a}\ \Delta} \qquad \frac{\Omega\ [\mathcal{R}]\ \Delta}{\Omega\ \underleftarrow{a}\ [\mathcal{R}]\ \Delta\ \underleftarrow{a}}$$

Processes should be equivalent only if they have the same input/output behavior. In this setting, there is a single type of observable behavior: output of outward-directed messeges. and input of inward-directed messeges.

DEFINITION 7.2. A *rewriting bisimulation*, $\mathcal{R}$, is a symmetric binary relation among contexts that satisfies the following conditions.
Output bisimulation If $\Omega \mathrel{\mathcal{R}} \Longrightarrow \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$, then $\Omega \Longrightarrow (\underleftarrow{\Delta}'_L \mathcal{R} \underrightarrow{\Delta}'_R)\ \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$.
Input bisimulation If $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R\ (\underrightarrow{\Delta}_L \mathcal{R} \underleftarrow{\Delta}_R) \Longrightarrow \Delta'$, then $\underrightarrow{\Delta}_L\ \Omega\ \underleftarrow{\Delta}_R \Longrightarrow \mathcal{R}\ \Delta'$.
Rewriting bisimilarity, $\cong$, is the largest rewriting bisimulation.

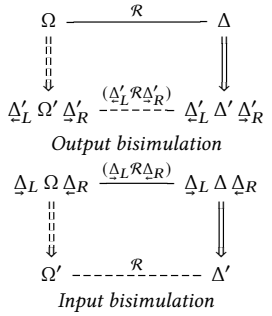


Figure 7.3: Rewriting bisimulation conditions, in diagrams

These are very strong conditions – arbitrary traces and quantify over all output/input contexts.

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 $\underrightarrow{\Delta}'_R$ and $\underrightarrow{\Delta}_L$ and $\underleftarrow{\Delta}_R$, respectively, are empty:

THEOREM 7.4. *If $\mathcal{R}$ is a rewriting bisimulation, then $\mathcal{R}$ satisifies:*
Reduction bisimulation *If $\Omega \mathrel{\mathcal{R}} \Longrightarrow \Delta'$, then $\Omega \Longrightarrow \mathcal{R}\ \Delta'$.*

Because rewriting bisimilarity is defined coinductively, with very strong conditions, the **??** itself is [...].

- The contexts $\underleftarrow{a}\ /\ \underleftarrow{a}$ and $\cdot$ are *not* bisimilar. Suppose, for the sake of contradiction, that they are bisimilar and so, framing $\underleftarrow{b}$ onto the right, we have $(\underleftarrow{a}\ /\ \underleftarrow{a})\ \underleftarrow{b}\ (\cong \underleftarrow{b})\ \underleftarrow{b}$. Composing the input and ouput bisimulation conditions, $(\underleftarrow{a}\ /\ \underleftarrow{a})\ \underleftarrow{b} \Longrightarrow (\underleftarrow{b} \cong)\ \underleftarrow{b}$ must follow. However, this is impossible: $(\underleftarrow{a}\ /\ \underleftarrow{a})\ \underleftarrow{b}$ is irreducible and does not expose $\underleftarrow{b}$ at its left end. Therefore, $\underleftarrow{a}\ /\ \underleftarrow{a}$ and $\cdot$ *cannot* be bisimilar.

- The contexts $\underrightarrow{a}$ and $\underrightarrow{a} \,\&\, \underrightarrow{b}$ are not bisimilar. The context $\underrightarrow{a} \,\&\, \underrightarrow{b}$ can output $\underrightarrow{b}$ at its right end: $\underrightarrow{a} \,\&\, \underrightarrow{b} \longrightarrow \underrightarrow{b}$. But $\underrightarrow{a}$ cannot simulate that output: the output bisimulation condition demands $\underrightarrow{a} \Longrightarrow (\cong \underrightarrow{b})\ \underrightarrow{b}$, which is impossible.

Now we would like to confirm our earlier intuition about the equivalence of $\underrightarrow{a}\ (\underrightarrow{a}\ /\ \underrightarrow{b})$ and $\underrightarrow{b}$ by proving that $\underrightarrow{a}\ (\underrightarrow{a}\ /\ \underrightarrow{b}) \cong \underrightarrow{b}$. 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 [...].

For instance,

Input bisimulation $\underrightarrow{\Delta}_L\ \underrightarrow{a}\ (\underrightarrow{a} \backslash \underrightarrow{b})\ \underleftarrow{\Delta}_R \Longrightarrow \Delta'$ implies $\underrightarrow{\Delta}_L\ \underrightarrow{b}\ \underleftarrow{\Delta}_R \cong \Delta'$; and $\underrightarrow{\Delta}_L\ \underrightarrow{b}\ \underleftarrow{\Delta}_R \Longrightarrow \Delta'$ implies $\underrightarrow{\Delta}_L\ \underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})\ \underleftarrow{\Delta}_R \cong \Delta'$; and

Output bisimulation $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b}) \Longrightarrow \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$ implies $\underrightarrow{b} \Longrightarrow (\underleftarrow{\Delta}'_L \cong \underrightarrow{\Delta}'_R)\ \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$; and $\underrightarrow{b} \Longrightarrow \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$ implies $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b}) \Longrightarrow (\underleftarrow{\Delta}'_L \cong \underrightarrow{\Delta}'_R)\ \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$.

In this small example, it is possible to imagine tediously proving these statements – after all, there are not that many traces involving $\underrightarrow{a}\ (\underrightarrow{a} \setminus \underrightarrow{b})$. However, in general, a proof technique for rewriting bisimilarity is needed.

Simple examples of bisimilar (or non-bisimilar) contexts

- $\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a} \not\cong \cdot$ because input bisimulation followed by output bisimulation demands that $\underset{\rightarrow}{b}\,(\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a}) \Longrightarrow (\cong \underset{\rightarrow}{b})\ \underset{\rightarrow}{b}$, which is impossible because $\underset{\rightarrow}{b}\,(\underset{\leftarrow}{a} \,/\, \underset{\leftarrow}{a})$ has no nontrivial reductions and does not expose $\underset{\rightarrow}{b}$ at its right.
- $\underset{\rightarrow}{a} \,\&\, \underset{\rightarrow}{b} \not\cong \underset{\rightarrow}{a}$ because $\underset{\rightarrow}{a} \Longrightarrow (\cong \underset{\rightarrow}{b})\ \underset{\rightarrow}{b}$ is impossible.
- [$\underset{\rightarrow}{a} \,\&\, \top \cong \underset{\rightarrow}{a}$, but only because rewriting is (weakly) focused.]
- $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b}) \cong \underset{\rightarrow}{b}$ intuitively because $\underset{\rightarrow}{a}\,(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{b})$ has no input transitions and reduces to $\underset{\rightarrow}{b}$. Need a proof technique to establish this.
- $\underset{\rightarrow}{a} \setminus (\underset{\rightarrow}{c} \,/\, \underset{\rightarrow}{b}) \cong (\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) \,/\, \underset{\rightarrow}{b}$ intuitively because the two propositions are logically equivalent. Both have the same input transitions. Also, $\underset{\rightarrow}{a} \setminus \uparrow\downarrow(\underset{\rightarrow}{c} \,/\, \underset{\leftarrow}{b}) \cong \uparrow\downarrow(\underset{\rightarrow}{a} \setminus \underset{\rightarrow}{c}) \,/\, \underset{\leftarrow}{b}$.

7.7.1 *Labeled bisimilarity: A proof technique for rewriting bisimilarity*

In 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. 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.

In this **??**, we follow that strategy and develop *labeled bisimilarity* as a sound and complete proof technique for rewriting bisimilariy. Like the π-calculus analogues, labeled bisimilarity is more tractable than rewriting bisimilarity because it uses labeled input transitions in place of [full] rewriting sequences.

Definition 7.3. A *labeled bisimulation*, $\mathcal{R}$, is a symmetric binary relation [among contexts] that satisfies the following conditions.

Immediate output bisimulation If $\Omega\ \mathcal{R}\ \Delta = \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$, then $\Omega \Longrightarrow (\underset{\leftarrow}{\Delta}'_L \mathcal{R} \underset{\rightarrow}{\Delta}'_R)\ \Delta$.

Immediate input bisimulation If $\Omega\ \mathcal{R}\ \Delta$ and $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$, then $\underset{\rightarrow}{\Delta}_L\ \Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow \mathcal{R}\ \Delta'$.

Reduction bisimulation If $\Omega\ \mathcal{R} \longrightarrow \Delta'$, then $\Omega \Longrightarrow \mathcal{S}\ \Delta'$.

Emptiness bisimulation If $\Omega\ \mathcal{R}\ \cdot$, then: $\underset{\rightarrow}{\Delta}\ \Omega \Longrightarrow (\mathcal{R}\underset{\rightarrow}{\Delta})\ \underset{\rightarrow}{\Delta}$ for all $\underset{\rightarrow}{\Delta}$; and $\Omega\ \underset{\leftarrow}{\Delta} \Longrightarrow (\underset{\leftarrow}{\Delta}\mathcal{R})\ \underset{\leftarrow}{\Delta}$ for all $\underset{\leftarrow}{\Delta}$.

Labeled bisimilarity is the largest labeled bisimulation.

The emptiness bisimulation condition is necessary to [...]. Notice that it is equivalent to $\Omega\ \mathcal{R}\ \cdot$ implies $\Omega \Longrightarrow \cdot$. A similar condition appears in **Deng+:LINEARITY12**.

Theorem 7.5 (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 conditions are trivial instances of the output and

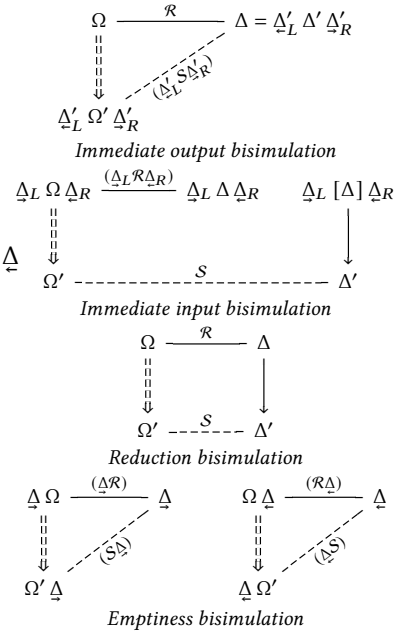


Figure 7.4: Labeled bisimulation conditions, in diagrams

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. □

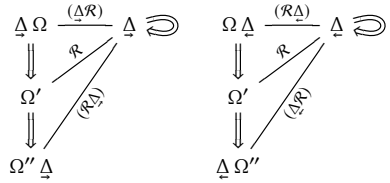


Figure 7.5: Emptiness bisimulation property as a consequence of input and output bisimulation properties

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 $\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b})\ \mathcal{R}\ (\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b}$ and $\underrightarrow{c}\ \mathcal{R}\ \underrightarrow{c}$ and $(\cdot)\ \mathcal{R}\ (\cdot)$. The relation $\mathcal{R}$ is a labeled bisimulation, but it does not satisfy the input bisimulation condition, because $\underrightarrow{a}\,(\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b}))\ (\underrightarrow{a}\mathcal{R})\ \underrightarrow{a}\,((\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b})$ does not imply $\underrightarrow{a}\,(\underrightarrow{a} \setminus (\underrightarrow{c} \,/\, \underrightarrow{b})) \Longrightarrow \mathcal{R}\ \underrightarrow{a}\,((\underrightarrow{a} \setminus \underrightarrow{c}) \,/\, \underrightarrow{b})$, and so is not a rewriting bisimulation.

However, a slightly weaker statement *is* true: a labeled bisimulation is contained within *some* rewriting bisimulation. Specifically, if $\mathcal{R}$ is a labeled bisimulation, then its input contextual closure, $[\mathcal{R}]$, is such a rewriting bisimulation. Fortunately, this will be enough to prove that labeled bisimilarity is sound.

Definition 7.4. A symmetric binary relation $\mathcal{R}$ *(labeled-)progresses* to binary relation $\mathcal{S}$ if the two relations satisfy the following conditions.

Immediate output bisimulation If $\Omega\ \mathcal{R}\ \Delta = \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$, then $\Omega \Longrightarrow (\underleftarrow{\Delta}'_L\mathcal{S}\underrightarrow{\Delta}'_R)\ \Delta$.

Immediate input bisimulation If $\Omega\ \mathcal{R}\ \Delta$ and $\underrightarrow{\Delta}_L\,[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$, then $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R \Longrightarrow \mathcal{S}\ \Delta'$.

Reduction bisimulation If $\Omega\ \mathcal{R} \longrightarrow \Delta'$, then $\Omega \Longrightarrow \mathcal{S}\ \Delta'$.

Emptiness bisimulation If $\Omega\ \mathcal{R}\ \cdot$, then: $\underrightarrow{\Delta}\,\Omega \Longrightarrow (\mathcal{S}\underrightarrow{\Delta})\ \underrightarrow{\Delta}$ for all $\underrightarrow{\Delta}$; and $\Omega\,\underleftarrow{\Delta} \Longrightarrow (\underleftarrow{\Delta}\mathcal{S})\ \underleftarrow{\Delta}$ for all $\underleftarrow{\Delta}$.

Notice that the labeled bisimulations are exactly those relations that progress to themselves.

Lemma 7.6. *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 $\mathcal{R} \cup \mathcal{S}$ satisfies the conditions required of a labeled bisimulation. If $\Omega\ (\mathcal{R} \cup \mathcal{S})\ \Delta$ because Ω and Δ are $\mathcal{R}$-related □

Next, we show that

Lemma 7.7. *If $\mathcal{R}$ is a labeled bisimulation, then so are $(\underrightarrow{a}\mathcal{R}) \cup \mathcal{R}$ and $(\mathcal{R}\underrightarrow{a}) \cup \mathcal{R}$, for all $\underrightarrow{a}$.*

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 $(\mathcal{R}\underrightarrow{a}) \cup \mathcal{R}$ is symmetric.

According to lemma 7.6, 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 $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\ \Delta = \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$; we must show that $\Omega \Longrightarrow\big(\underset{\leftarrow}{\Delta}'_L((\underset{\rightarrow}{a}\mathcal{R}) \cup \mathcal{R})\underset{\rightarrow}{\Delta}'_R\big)\ \Delta$. Because the input atom $\underset{\rightarrow}{a}$ cannot be unified with the output atoms $\underset{\leftarrow}{\Delta}'_L$, the context $\underset{\leftarrow}{\Delta}'_L$ 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 $\underset{\rightarrow}{a}$ and deduce that $\Omega\ \big(\underset{\rightarrow}{a}(\Longrightarrow(\mathcal{R}\underset{\rightarrow}{\Delta}'_R))\big)\ \Delta$. Reduction is closed under framing, so we conclude that $\Omega \Longrightarrow((\underset{\rightarrow}{a}\mathcal{R})\underset{\rightarrow}{\Delta}'_R)\ \Delta$, as required.

$$\Omega = \underset{\rightarrow}{a}\,\Omega_0 \xrightarrow{(\underset{\rightarrow}{a}\mathcal{R})} \underset{\rightarrow}{a}\,\Delta'_0\,\underset{\rightarrow}{\Delta}'_R = \Delta \qquad \underset{\rightarrow}{a}\,\Omega'_0 \quad ((\underset{\rightarrow}{a}\mathcal{R})\underset{\rightarrow}{\Delta}'_R)$$

- Consider the case in which Δ' is empty – that is, the case in which $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\ \Delta = \underset{\rightarrow}{\Delta}'_R = \underset{\rightarrow}{a}\,\underset{\rightarrow}{\Delta}''_R$ for some $\underset{\rightarrow}{\Delta}''_R$. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate output bisimulation property after framing off $\underset{\rightarrow}{a}$ and deduce that $\Omega\ (\underset{\rightarrow}{a}(\Longrightarrow(\mathcal{R}\underset{\rightarrow}{\Delta}''_R)))\ \underset{\rightarrow}{\Delta}'_R$. Reduction is closed under framing, so $\Omega \Longrightarrow((\underset{\rightarrow}{a}\mathcal{R})\underset{\rightarrow}{\Delta}''_R)\ \underset{\rightarrow}{\Delta}'_R$. After framing off $\underset{\rightarrow}{\Delta}''_R$, we may subsequently appeal to the emptiness bisimulation property of $\mathcal{R}$ and deduce that $\Omega \Longrightarrow((\Longrightarrow(\mathcal{R}\underset{\rightarrow}{a}))\underset{\rightarrow}{\Delta}''_R)\ \underset{\rightarrow}{\Delta}'_R$. Once again, reduction is closed under framing, so we conclude that $\Omega \Longrightarrow(\mathcal{R}\underset{\rightarrow}{\Delta}'_R)\ \Delta$, as required.

$$\Omega_0 \xrightarrow{\mathcal{R}} \Delta'_0\,\underset{\rightarrow}{\Delta}'_R \qquad \Omega'_0 \quad (\mathcal{R}\underset{\rightarrow}{\Delta}'_R)$$

$$\Omega = \underset{\rightarrow}{a}\,\Omega_0 \xrightarrow{(\underset{\rightarrow}{a}\mathcal{R})} \underset{\rightarrow}{a}\,\underset{\rightarrow}{\Delta}''_R = \underset{\rightarrow}{\Delta}'_R \qquad \underset{\rightarrow}{a}\,\Omega'_0 \quad ((\underset{\rightarrow}{a}\mathcal{R})\underset{\rightarrow}{\Delta}''_R) \qquad \Omega''_0\,\underset{\rightarrow}{a} \quad ((\mathcal{R}\underset{\rightarrow}{a})\underset{\rightarrow}{\Delta}''_R)$$

Immediate input bisimulation Assume that $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\ \Delta$ and $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$; we must show that $\underset{\rightarrow}{\Delta}_L\ \Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow((\underset{\rightarrow}{a}\mathcal{R}) \cup \mathcal{R})\ \Delta'$. According to lemma 7.3, there are two cases: either $\underset{\rightarrow}{a}$ satisfies an input demand, or it does not participate in the given input transition.

- Consider the case in which $\underset{\rightarrow}{a}$ does participate in the input transition – that is, the case in which $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\ \underset{\rightarrow}{a}\,\Delta_0 = \Delta$ and $\underset{\rightarrow}{\Delta}_L\ \underset{\rightarrow}{a}\ [\Delta_0]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$, for some Δ_0. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its immediate input bisimulation property and deduce that $\underset{\rightarrow}{\Delta}_L\ \Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow\mathcal{R}\ \Delta'$, as required.
- Consider the case in which $\underset{\rightarrow}{a}$ does not participate in the transition – that is, the case in which $\underset{\rightarrow}{\Delta}_L$ is empty and $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\ \underset{\rightarrow}{a}\,\Delta_0 = \Delta$ and $[\Delta_0]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'_0$ and $\Delta' = \underset{\rightarrow}{a}\,\Delta'_0$, 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 $\underset{\rightarrow}{a}$ and deduce that $\Omega\ \underset{\leftarrow}{\Delta}_R\ (\underset{\rightarrow}{a}(\Longrightarrow\mathcal{R}))\ \Delta'$. Reduction is closed under framing, so we conclude that $\Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow(\underset{\rightarrow}{a}\mathcal{R})\ \Delta'$, as required.

Reduction bisimulation Assume that $\Omega\ (\underset{\rightarrow}{a}\mathcal{R})\longrightarrow \Delta'$; we must show that $\Omega \Longrightarrow((\underset{\rightarrow}{a}\mathcal{R}) \cup \mathcal{R})\ \Delta'$. 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 $\Omega\ (\underset{\rightarrow}{a}(\mathcal{R}\longrightarrow))\ \Delta'$. Because $\mathcal{R}$ is a labeled bisimulation, we may appeal to its reduction bisimulation property after framing off $\underset{\rightarrow}{a}$ and deduce that $\Omega\ (\underset{\rightarrow}{a}(\Longrightarrow\mathcal{R}))\ \Delta'$. Reduc-

tion 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 $\underrightarrow{a}$. □

Lemma 7.8. *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 7.7, 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 $\bigcup_{n=0}^{\infty}\mathcal{S}_n = [\mathcal{R}]$. □

Theorem 7.9. *If $\mathcal{R}$ is a labeled bisimulation, then rewriting bisimilarity contains $\mathcal{R}$.*

Proof. Let $\mathcal{R}$ be a labeled bisimulation. By lemma 7.8, 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 $\Omega\ [\mathcal{R}]\ \Delta$ and $\underrightarrow{\Delta}_L\,[\Delta]\,\underleftarrow{\Delta}_R \longrightarrow \Delta'$; we must show that $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R \Longrightarrow[\mathcal{R}]\ \Delta'$.

By ??, the given input transition gives rise to a reduction: $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R\ (\underrightarrow{\Delta}_L[\mathcal{R}]\underleftarrow{\Delta}_R)\longrightarrow \Delta'$. Because $[\mathcal{R}]$ is input contextual, we deduce that $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R\ [\mathcal{R}]\longrightarrow \Delta'$. As a labeled bisimulation, $[\mathcal{R}]$ satisfies the reduction bisimulation property, so we conclude that $\underrightarrow{\Delta}_L\,\Omega\,\underleftarrow{\Delta}_R \Longrightarrow[\mathcal{R}]\ \Delta'$, as required. □

Corollary 7.10. *Labeled bisimilarity is sound and complete with respect to rewriting bisimilarity.*

As a simple example of this [labeled bisimilarity] proof technique, we shall now establish that $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b})$ and $\underrightarrow{b}$ are rewriting-bisimilar contexts. Let $\mathcal{R}$ be the least symmetric binary relation for which $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b})\ \mathcal{R}\ \underrightarrow{b}$ and $\underrightarrow{b}\ \mathcal{R}\ \underrightarrow{b}$ and $\cdot\ \mathcal{R}\ \cdot$ hold. The relation $\mathcal{R}$ is a labeled bisimulation:

- The immediate output bisimulation condition holds because $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b})$ can simulate $\underrightarrow{b}$'s output of $\underrightarrow{b}$ (with $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b}) \longrightarrow (\mathcal{R}\underrightarrow{b})\ \underrightarrow{b}$) and the former makes no immediate outputs of its own.
- The immediate input bisimulation condition holds vacuously for [the relation] $\mathcal{R}$ because neither $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b})$ nor $\underrightarrow{b}$ accept any inputs on either side.
- The reduction bisumulation condition holds because $\underrightarrow{b}$ can simulate the reduction $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b}) \longrightarrow \underrightarrow{b}$ trivially (with $\underrightarrow{b} \Longrightarrow \mathcal{R}\ \underrightarrow{b}$).
- The emptiness bisimulation condition holds trivially: $\underrightarrow{\Delta} \Longrightarrow (\mathcal{R}\underrightarrow{\Delta})\ \underrightarrow{\Delta}$ for all $\underrightarrow{\Delta}$ because $\cdot\ \mathcal{R}\ \cdot$, and symmetrically for all $\underleftarrow{\Delta}$.

We may conclude from the above proof technique (theorem 7.9) that $\mathcal{R}$ is contained within rewriting bisimilarity and that $\underrightarrow{a}\,(\underrightarrow{a} \setminus \underrightarrow{b})$ and $\underrightarrow{b}$ are indeed bisimilar.

We can similarly prove that $\underrightarrow{a} \setminus (\underleftarrow{c} \,/\, \underrightarrow{b})$ and $(\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b}$ are rewriting-bisimilar by showing that the least symmetric relation $\mathcal{R}$ such that $\underrightarrow{a} \setminus (\underleftarrow{c} \,/\, \underrightarrow{b})\ \mathcal{R}\ (\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b}$ and $\underleftarrow{c}\ \mathcal{R}\ \underleftarrow{c}$ and $\cdot\ \mathcal{R}\ \cdot$ is a labeled bisimulation.

Somewhat surprisingly, even $\underrightarrow{a} \setminus \uparrow\downarrow(\underleftarrow{c} \,/\, \underrightarrow{b})$ and $\uparrow\downarrow(\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b}$ are bisimilar. Once again, this can be proved by constructing an appropriate label bisimulation, namely the least symmetric relation $\mathcal{R}$ such that: $\underrightarrow{a} \setminus \uparrow\downarrow(\underleftarrow{c} \,/\, \underrightarrow{b})\ \mathcal{R}\ \uparrow\downarrow(\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b}$ and $\underleftarrow{c} \,/\, \underrightarrow{b}\ \mathcal{R}\ \underrightarrow{a}\,(\uparrow\downarrow(\underrightarrow{a} \setminus \underleftarrow{c}) \,/\, \underrightarrow{b})$ and $\big(\underrightarrow{a} \setminus \uparrow\downarrow(\underleftarrow{c} \,/\, \underrightarrow{b})\big)\ \underrightarrow{b}\ \mathcal{R}\ \underrightarrow{a} \setminus \underleftarrow{c}$ and $\underleftarrow{c}\ \mathcal{R}\ \underleftarrow{c}$ and $\cdot\ \mathcal{R}\ \cdot$.

7.7.2 *A simple up-to proof technique: Reflexivity*

As a slight enhancement of the above proof technique,

Lemma 7.11. *The identity relation is a labeled bisimulation.*

Proof. Each of the labeled bisimulation conditions is trivially true of the identity relation. □

Let us call a relation $\mathcal{R}$ a labeled bisimulation *up to reflexivity* if $\mathcal{R}$ progresses to its reflexive closure, $\mathcal{R}^{=}$.

Theorem 7.12. *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. notice that Because the identity relation is a labeled bisimulation (??), it follows from ?? that $\mathcal{R}^{=}$, the reflexive closure of $\mathcal{R}$, is a labeled bisimulation. By ??, we may conclude that rewriting bisimilarity contains $\mathcal{R}^{=}$ and hence $\mathcal{R}$. □

7.8

Rewriting bisimilarity satisfies the usual properties expected of a notion of bisimilarity: it is a congruence [...].

THEOREM 7.13. *Rewriting bisimilarity is an equivalence relation.*

Proof. Equality of contexts can be shown to be a bisimulation, so rewriting bisimilarity is reflexive. Rewriting bisimilarity is symmetric, by definition. The relation $\cong^2$ can be shown to be a bisimulation, so rewriting bisimilarity is transitive. □

LEMMA 7.14. *Rewriting bisimilarity contains* $(A^+\cong)$ *and* $(\cong A^+)$*, for all propositions* A^+*.*

Proof. By induction over the structure of A^+. We show only the proof for $(A^+\cong)$; the proof for $(\cong A^+)$ is symmetric.

According to **??**, it suffices to show that $(A^+\cong) \cup \cong$ is a labeled bisimulation. By **??**, we need only show that $(A^+\cong)$ progresses to $(A^+\cong) \cup \cong$. Many of the cases follow the pattern laid out in the proof of **??**, substituting A^+ for $\underset{\rightarrow}{a}$; we show only the new cases.

Immediate output bisimulation Assume that $\Omega\ (A^+\cong)\ \Delta = \underset{\leftarrow}{\Delta}'_L\ \Delta'\ \underset{\rightarrow}{\Delta}'_R$; we must show that $\Omega \Longrightarrow\big(\underset{\leftarrow}{\Delta}'_L((A^+\cong) \cup \cong)\underset{\rightarrow}{\Delta}'_R\big)\ \Delta$.

Unlike in the proof of **??**, here it is possible that $A^+ = \underset{\leftarrow}{a}$ with $\underset{\leftarrow}{\Delta}'_L$ nonempty: $\underset{\leftarrow}{\Delta}'_L = \underset{\leftarrow}{a}\,\underset{\leftarrow}{\Delta}''_L$, for some $\underset{\leftarrow}{\Delta}''_L$. Because $\cong$ is a labeled bisimulation (**??**), we may appeal to its immediate output bisimulation property after framing $\underset{\leftarrow}{a}$ and deduce that $\Omega\ \big(\underset{\leftarrow}{a}(\Longrightarrow(\underset{\leftarrow}{\Delta}''_L\cong\underset{\rightarrow}{\Delta}'_R))\big)\ \Delta$. Reduction is closed under framing, so $\Omega \Longrightarrow(\underset{\leftarrow}{\Delta}'_L\cong\underset{\rightarrow}{\Delta}'_R)\ \Delta$, as required.

The other cases follow the pattern laid out in the proof of **??**.

Immediate input bisimulation Assume that $\Omega\ (A^+\cong)\ \Delta$ and $\underset{\rightarrow}{\Delta}_L\ [\Delta]\ \underset{\leftarrow}{\Delta}_R \longrightarrow \Delta'$; we must show that $\underset{\rightarrow}{\Delta}_L\ \Omega\ \underset{\leftarrow}{\Delta}_R \Longrightarrow\big((A^+\cong) \cup \cong\big)\ \Delta'$.

All cases here follow the pattern laid out in the proof of **??**.

Reduction bisimulation Assume that $\Omega\ (A^+\cong)\longrightarrow\ \Delta'$; we must show that $\Omega \Longrightarrow((A^+\cong) \cup \cong)\ \Delta'$.

As in the proof of **??**, we distinguish cases based on the origin of the reduction. Here, there are three new cases because the reduction might originate from A^+; the other cases follow the pattern laid out in the proof of **??**.

- Consider the case in which $A^+ = A^+_1 \bullet A^+_2$ and $\Omega\ (A^+\cong)\ (A^+_1 \bullet A^+_2)\ \Delta_0 \longrightarrow A^+_1\ A^+_2\ \Delta_0 = \Delta'$ for some Δ_0. Notice that $\Omega \longrightarrow(A^+_1(A^+_2\cong))\ \Delta'$. By appealing to the inductive hypothesis for A^+_2, we deduce that rewriting bisimilarity contains $(A^+_2\cong)$, and so $\Omega \longrightarrow(A^+_1\cong)\ \Delta'$. Similar reasoning for A^+_1 allows us to conclude that $\Omega \longrightarrow\cong\ \Delta'$, as required.

- Consider the case in which $A^+ = \mathbf{1}$ and $\Omega\ (A^+\cong)\ \mathbf{1}\,\Delta' \longrightarrow \Delta'$. Notice that $\Omega \longrightarrow\cong \Delta'$, as required.
- Consider the case in which $A^+ = A_0^-$ and $\Omega\ \left(A^+\cong\right)\ \downarrow A_0^- \underleftarrow{\Delta}_L \Delta_0' \longrightarrow C^+\,\Delta_0' = \Delta'$ because $[A_0^-]\,\underleftarrow{\Delta}_L \Vdash C^+$. Because $\cong$ is a labeled bisimulation (??), we may appeal to immediate output bisimulation property after framing off $\downarrow A_0^-$ and deduce that $\Omega\ \left(A^+(\Longrightarrow(\underleftarrow{\Delta}_L\cong))\right)\ \downarrow A_0^- \underleftarrow{\Delta}_L \Delta_0'$. Reduction is closed under framing, so $\Omega \Longrightarrow\left((\downarrow A^- \underleftarrow{\Delta}_L)\cong\right)\ \downarrow A^- \underleftarrow{\Delta}_L \Delta_0'$. We can insert the reduction $\downarrow A^- \underleftarrow{\Delta}_L \longrightarrow C^+$ and arrive at $\Omega \Longrightarrow(C^+\cong)\ C^+\,\Delta_0' = \Delta'$. The proposition C^+ is a subformula of $\downarrow A^-$, so, by the inductive hypothesis, $(C^+\cong)$ is contained within $\cong$. It follows that $\Omega \Longrightarrow\cong \Delta'$, as required.

Emptiness bisimulation Assume that $\Omega\ (A^+\cong)\ \cdot$. As before, this is impossible. □

THEOREM 7.15. *If $\Omega \cong \Delta$, then $\Omega_L\,\Omega\,\Omega_R \cong \Omega_L\,\Delta\,\Omega_R$ for all Ω_L and Ω_R.*

Proof. By induction on the structures of Ω_L and Ω_R, appealing to ??. □

THEOREM 7.16 (Reduction closure). *Let $\mathcal{R}$ be a rewriting bisimulation. Then $\mathcal{R}$ is reduction-closed: $\Omega\ \mathcal{R}\Longrightarrow\ \Delta'$ implies $\Omega \Longrightarrow\mathcal{R}\ \Delta'$.*

Proof. Reduction closure follows immediately as the trivial instance of either the output or input bisimulation properties. □

7.8.1 *Counterexample*

This definition is too fine, ruling out desirable equivalences. For example, $e\,b_0 \ncong e$. Suppose, for the sake of deriving a contradiction, that $e\,b_0 \cong e$. Because $e\,b_0\,\underline{d} \Longrightarrow \underline{z}\,b_0'$, it follows from input bisimilarity that $e\,\underline{d} \Longrightarrow\cong^{-1} \underline{z}\,b_0'$. So either $\underline{z}\,b_0' \cong e\,\underline{d}$ or $\underline{z}\,b_0' \cong \underline{z}$. The former is impossible because $\underline{z}\,b_0'$ cannot produce $\underline{d}$ on the right[16] and so violates output bisimilarity.

[16] Nor, in fact, on the left.

The latter is also impossible. It has an output of $\underline{z}$ on the left of $\underline{z}\,b_0'$, from which output bisimilarity yields $b_0' \cong \cdot$. From input bisimilarity, $b_0'\,\underline{a} \cong \underline{a}$ follows, for any $\underline{a}$. And, that violates output bisimilarity because $b_0'\,\underline{a}$, 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).

7.9 *Example of rewriting bisimilarity: Nondeterministic finite automata*

As a more elaborate example of rewriting bisimilarity, we can return to the encoding of NFAs proposed in ??. Recall that

$$\hat{q} \triangleq$$

Like DFA states, NFA states that have equal encodings are bisimilar; unnlike DFA states, bisimilar NFA states do not have equal encodings. However, bisimilar NFA states do have encodings that are rewriting-bisimilar. In other words, the NFA encoding preserves bisimilarity: $q \sim s$ if, and only if, $\hat{q} \cong \hat{s}$.

[...]

Before proving this statement, we need a few **??**.

LEMMA 7.17. *For all states q:*

1. $\hat{q} \not\longrightarrow$.
2. *If $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow \Omega'$, then either $\underset{\rightarrow}{a}\,\hat{q} = \Omega'$ or $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a = \Omega'$ for some state q'_a that a-succeeds q.*
3. *If $\underset{\rightarrow}{\epsilon}\,\hat{q} \Longrightarrow \Omega'$, then either: $\underset{\rightarrow}{\epsilon}\,\hat{q} = \Omega'$; $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q) = \Omega'$; or q is a final state and $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q) \longrightarrow \cdot = \Omega'$.*

Proof. Part 1 is proved by inversion of a hypothetical rewriting of $\hat{q}$.

Part 2 is proved by inversion of the given rewriting sequence: If the rewriting sequence is nontrivial, it must be $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'_a \Longrightarrow \Omega'$ for some state q'_a that a-succeeds q; by part 1, we deduce that $\Omega' = \hat{q}'_a$. Otherwise, if the rewriting sequence is trivial, the desired result is immediate. □

These results hold only because ordered rewriting is weakly focused; under an unfocused rewriting framework, $\hat{q}$ would admit rewritings, such as $\hat{q} \Longrightarrow \underset{\leftarrow}{\epsilon} \setminus \hat{F}(q)$, and $\underset{\rightarrow}{a}\,\hat{q}$ would admit rewritings to contexts other than encodings of a-successors.

LEMMA 7.18. *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.*

Proof. Assume that $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow\cong \hat{q}'$. According to lemma 7.17, there are two cases: either *(i)* $\underset{\rightarrow}{a}\,\hat{q} \cong \hat{q}'$ or *(ii)* $\hat{q}'_a \cong \hat{q}'$ for some state q'_a that a-succeeds q. In the latter case, the desired result is immediate.

In the former case, because the underlying NFA is well-formed (**??**), 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 7.4), $\hat{q}'_a \cong\Longleftarrow \hat{q}'$. States are encoded by latent[17] propositions (lemma 7.17), and so we may conclude that, in fact, $\hat{q}'_a \cong \hat{q}'$. □

[17] **??**.

LEMMA 7.19. *If $\underset{\rightarrow}{\epsilon}\,\hat{q} \Longrightarrow\cong \hat{F}(s)$, then $q \in F$ if, and only if, $s \in F$.*

Proof. • Consider the case in which the trace is trivial – *i.e.*, $\underset{\rightarrow}{\epsilon}\,\hat{q} \cong \hat{F}(s)$. By definition of the encoding, $\hat{F}(q) \longleftarrow \underset{\rightarrow}{\epsilon}\,\hat{q} \cong \hat{F}(s)$. $\hat{F}(q) \cong\Longleftarrow \hat{F}(s)$

□

LEMMA 7.20. *If $\hat{F}(q) \Longrightarrow\cong \hat{F}(s)$, then $q \in F$ if, and only if, $s \in F$.*

Proof. Assume that $\hat{F}(q) \Longrightarrow\cong \hat{F}(s)$ and $q \notin F$. By inversion, The trace can only be the trivial one, so $\hat{F}(q) = \top$ and $\hat{F}(s)$ are bisimilar. Suppose, for the sake of contradiction, that $s \in F$ and so $\hat{F}(s) = \mathbf{1}$. Then $\top \cong \mathbf{1}$; hence,

$\underset{\rightarrow}{a}\,\top \Longrightarrow\cong \underset{\rightarrow}{a}$ follows from the input bisimilarity property. But output bisimilarity implies $\underset{\rightarrow}{a}\,\top \Longrightarrow(\cong\underset{\rightarrow}{a})\ \underset{\rightarrow}{a}$, which is impossible because $\underset{\rightarrow}{a}\,\top$ cannot produce $\underset{\rightarrow}{a}$ at its right end. □

THEOREM 7.21 (NFA adequacy). *Let $\mathcal{A} = (Q, ?, F)$ be an NFA over the input alphabet Σ. Then, for all states q, q', and s:*

1. *$q \sim s$ if, and only if, $\hat{q} \cong \hat{s}$.*
2. *$q \sim\xrightarrow{a}\sim q'$ if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \cong\longrightarrow\cong \hat{q}'$, for all input symbols $a \in \Sigma$. Moreover, $q \sim\xrightarrow{w}\sim q'$ if, and only if, $\underset{\rightarrow}{w}\,\hat{q} \cong\Longrightarrow\cong \hat{q}'$, for all finite words $w \in \Sigma^*$.*
3. *$q \in F$ if, and only if, $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \mathbf{1}$.*

Proof. Each part is proved in turn. The proof of part 2 depends on the proof of part 1.

1. 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 will show that the symmetric relation $\mathcal{R} = \{(\hat{q}, \hat{s}) \mid q \sim s\}$ is a labeled bisimulation up to reflexivity and, by theorem 7.12, is included in rewriting bisimilarity.

     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 propostion 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 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 $\underset{\rightarrow}{\epsilon}\,[\hat{s}] \longrightarrow \hat{F}(s)$. Because q and s are bisimilar states, $\hat{F}(q) = \hat{F}(s)$ (??). By definition of the encoding, $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q)$, and so, indeed, $\underset{\rightarrow}{\epsilon}\,\hat{q} \Longrightarrow\mathcal{R}^{=}\ \hat{F}(s)$, 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 latent propositions, and so $\hat{s}$ does not reduce (??).

     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 will 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 bisimilarity.

  Because rewriting bisimilarity is symmetric, so too is the relation $\mathcal{R}$.

  – 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 7.18 provides exactly what is needed: a state s'_a that a-succeeds s and has an encoding bisimilar to that of q'_a.

  – 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, $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q) = \mathbf{1}$. Because q and s have bisimilar encodings, it follows from input bisimilarity that $\underset{\rightarrow}{\epsilon}\,\hat{s} \Longrightarrow\!\cong \hat{F}(q)$. An appeal to **??** yields $\hat{F}(q) = \hat{F}(s)$, from which we deduce that s is also a final state.

2. We would like to prove that $q \sim\!\xrightarrow{w}\!\sim q'$ if, and only if, $\underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q} \cong\!\Longrightarrow\!\cong \hat{q}'$. Because bisimilar states have bisimilar encodings (part 1), because rewriting bisimilarity is left-congruent (**??**), reduction-closed (theorem 7.4), and transitive (**??**), and because NFA bisimilarity is reflexive (**??**), it suffices to show: (a) that $q \xrightarrow{w} q'$ implies $\underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$; and (b) that $\underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q} \Longrightarrow\!\cong \hat{q}'$ implies $q \xrightarrow{w}\!\sim q'$. We prove these two statements in turn.

   (a) That $q \xrightarrow{w} q'$ implies $\underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q} \Longrightarrow \hat{q}'$ can be proved by a straightforward induction over the structure of word w; we omit the details.

   (b) We shall prove that $\underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q} \Longrightarrow\!\cong \hat{q}'$ implies $q \xrightarrow{w}\!\sim q'$ by induction over the structure of word w.

   - Consider the case of the empty word, ϵ; we must show that $\hat{q} \Longrightarrow\!\cong \hat{q}'$ implies $q \sim q'$. Because states are encoded by latent propositions (lemma 7.17), the given trace can only be the trivial one; thus, $\hat{q} \cong \hat{q}'$. An appeal to part 1 allows us to conclude that $q \sim q'$.
   - Consider the case of a nonempty word, $a\,w$; we must show that $\underset{\rightarrow}{w}^{\mathrm{R}}\,\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow\!\cong \hat{q}'$ implies $q \xrightarrow{a}\!\xrightarrow{w}\!\sim q'$. By inversion, the given trace must begin by inputting a:

     $$\underset{\rightarrow}{w}^{\mathrm{R}}\,\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \underset{\rightarrow}{w}^{\mathrm{R}}\,\hat{q}'_a \Longrightarrow\!\cong \hat{q}'\,,$$

     where q'_a is an a-successor of state q. An appeal to the inductive hypothesis on the trace's tail yields $q'_a \xrightarrow{w}\!\sim q'$, and so the NFA admits $q \xrightarrow{a}\!\xrightarrow{w}\!\sim q'$, as required.

Additionally, we would like to prove an apparently stronger statement: $q \sim\!\xrightarrow{a}\!\sim q'$ if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \cong\!\longrightarrow\!\cong \hat{q}'$. From the above proof involving finite words, we know that $q \sim\!\xrightarrow{a}\!\sim q'$ if, and only if, $\underset{\rightarrow}{a}\,\hat{q} \cong\!\Longrightarrow\!\cong \hat{q}'$. Therefore, it suffices to show that the multi-step $\underset{\rightarrow}{a}\,\hat{q} \cong\!\Longrightarrow\!\cong \hat{q}'$ is equivalent to the single-step $\underset{\rightarrow}{a}\,\hat{q} \cong\!\longrightarrow\!\cong \hat{q}'$.

- To prove the left-to-right direction, begin with the multi-step assumption that $\underset{\rightarrow}{a}\,\hat{q} \cong\!\Longrightarrow\!\cong \hat{q}'$. Because rewriting bisimilarity is reduction-closed (theorem 7.4) and transitive (??), $\underset{\rightarrow}{a}\,\hat{q} \Longrightarrow\!\cong \hat{q}'$. By lemma 7.18, there exists a state q'_a that a-succeeds q such that $\hat{q}'_a \cong \hat{q}'$. Then, by definition of the encoding (and reflexivity of rewriting bisimilarity (??)), it follows that $\underset{\rightarrow}{a}\,\hat{q} \cong\!\longrightarrow\!\cong \hat{q}'$, as required.

- The right-to-left direction is trivial because a single step is a particular form of trace.

3. We shall prove that the final states are exactly those states q such that $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \mathbf{1}$.

   - Let q be a final state; accordingly, $\hat{F}(q) = \mathbf{1}$. There exists, by definition of the encoding, a trace $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q) = \mathbf{1}$.

   - Assume that $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \mathbf{1}$. The only step possible from $\underset{\rightarrow}{\epsilon}\,\hat{q}$ is $\underset{\rightarrow}{\epsilon}\,\hat{q} \longrightarrow \hat{F}(q)$, and so $\hat{F}(q) = \mathbf{1}$. It follows that q is a final state. □

7.10 *Example of rewriting bisimilarity: Binary counter*

For a further application of rewriting bisimilarity, we can revisit the binary counter of ??. Under a message-passing interpretation of ordered rewriting, the definitions of e, b_0, b_1, and b'_0 are the same as in ??, but with directions consistently assigned to the uninterpreted, positive atomic propositions:

$$
\begin{aligned}
e &\triangleq (e \bullet b_1 \;/\; \underset{\leftarrow}{i}) \;\&\; (\underset{\rightarrow}{z} \;/\; \underset{\leftarrow}{d}) \\
b_0 &\triangleq (\uparrow\!\downarrow b_1 \;/\; \underset{\leftarrow}{i}) \;\&\; (\underset{\leftarrow}{d} \bullet b'_0 \;/\; \underset{\leftarrow}{d}) \\
b'_0 &\triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \;\&\; (\underset{\rightarrow}{s} \setminus b_1 \bullet \underset{\rightarrow}{s}) \\
b_1 &\triangleq (\underset{\leftarrow}{i} \bullet b_0 \;/\; \underset{\leftarrow}{i}) \;\&\; (b_0 \bullet \underset{\rightarrow}{s} \;/\; \underset{\leftarrow}{d})
\end{aligned}
$$

The increment and decrement messages, $\underset{\leftarrow}{i}$ and $\underset{\leftarrow}{d}$, are left-directed, whereas the zero and successor messages, $\underset{\rightarrow}{z}$ and $\underset{\rightarrow}{s}$, are right-directed.

$$\frac{}{e \approx_{\mathrm{V}} 0} \qquad \frac{\Omega \approx_{\mathrm{V}} n}{\Omega\, b_0 \approx_{\mathrm{V}} 2n} \qquad \frac{\Omega \approx_{\mathrm{V}} n}{\Omega\, b_1 \approx_{\mathrm{V}} 2n+1}$$

$$\frac{}{e \approx_{\mathrm{I}} 0} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_0 \approx_{\mathrm{I}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, b_1 \approx_{\mathrm{I}} 2n+1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, \underset{\leftarrow}{i} \approx_{\mathrm{I}} n+1}$$

$$\frac{}{e \bullet b_1 \approx_{\mathrm{I}} 1} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(i \bullet b_0) \approx_{\mathrm{I}} 2(n+1)}$$

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, \underset{\leftarrow}{d} \approx_{\mathrm{D}} n} \qquad \frac{}{\underset{\rightarrow}{z} \approx_{\mathrm{D}} 0} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\, \underset{\rightarrow}{s} \approx_{\mathrm{D}} n+1} \qquad \frac{\Omega \approx_{\mathrm{D}} n}{\Omega\, b_0' \approx_{\mathrm{D}} 2n}$$

$$\frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(\underset{\leftarrow}{d} \bullet b_0') \approx_{\mathrm{D}} 2n} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_1 \bullet \underset{\rightarrow}{s}) \approx_{\mathrm{D}} 2n+2} \qquad \frac{\Omega \approx_{\mathrm{I}} n}{\Omega\,(b_0 \bullet \underset{\rightarrow}{s}) \approx_{\mathrm{D}} 2n+1}$$

THEOREM 7.22 (Big-step adequacy of increments). *If $\Omega \approx_{\mathrm{I}} n$, then:*

- $\Omega \Longrightarrow \approx_{\mathrm{V}} n'$ *if, and only if,* $n = n'$*; and*
- $\Omega \Longrightarrow \underset{\leftarrow}{\Omega_L'}\, \Omega' \,\underset{\rightarrow}{\Omega_R'}$ *only if* $\Omega' \approx_{\mathrm{I}} n$ *and both* $\underset{\leftarrow}{\Omega_L'}$ *and* $\underset{\rightarrow}{\Omega_R'}$ *are empty.*

THEOREM 7.23 (Big-step adequacy of decrements). *If $\Omega \approx_{\mathrm{D}} n$, then:*

- $\Omega \Longrightarrow \underset{\rightarrow}{z}$ *if* $n = 0$*;*
- $\Omega \Longrightarrow \Omega' \,\underset{\rightarrow}{s}$ *for some* $\Omega' \approx_{\mathrm{I}} n-1$ *if* $n > 0$*; and*
- $\Omega \Longrightarrow \underset{\leftarrow}{\Omega_L'}\, \Omega' \,\underset{\rightarrow}{\Omega_R'}$ *only if* $\underset{\leftarrow}{\Omega_L'} = \cdot$ *and either:*
  - $\Omega' \approx_{\mathrm{D}} n$ *and* $\underset{\rightarrow}{\Omega_R'} = \cdot$*;*
  - $n = 0$ *and* $\Omega' = \cdot$ *and* $\underset{\rightarrow}{\Omega_R'} = \underset{\rightarrow}{z}$*; or*
  - $n > 0$ *and* $\Omega' \approx_{\mathrm{I}} n-1$ *and* $\underset{\rightarrow}{\Omega_R'} = \underset{\rightarrow}{s}$*.*

Compare with:

THEOREM 7.24 (Big-step adequacy of decrements). *If $\Omega \approx_{\mathrm{D}} n$, then:*

- $\Omega \Longrightarrow \underset{\rightarrow}{z}$ *if* $n = 0$*;*
- $\Omega \Longrightarrow \Omega' \,\underset{\rightarrow}{s}$ *for some* $\Omega' \approx_{\mathrm{I}} n-1$ *if* $n > 0$*;*
- $\Omega \Longrightarrow \underset{\rightarrow}{z}$ *only if* $n = 0$*; and*
- $\Omega \Longrightarrow \Omega' \,\underset{\rightarrow}{s}$ *only if* $n > 0$ *and* $\Omega' \approx_{\mathrm{I}} n-1$*.*

7.10.1 *Counters with equal denotations are bisimilar*

States with equal denotations are bisimilar, and conversely, bimsimilar states have equal denotations.

THEOREM 7.25. *If $\Omega \approx_{\mathrm{I}} n$ and $\Delta \approx_{\mathrm{I}} n'$, then $\Omega \cong \Delta$ if, and only if, $n = n'$. Similarly, if $\Omega \approx_{\mathrm{D}} n$ 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

  $$\mathcal{R} = \{(\Omega, \Delta) | \exists n \in \mathbb{N}.\, (\Omega \approx_{\mathrm{I}} n) \wedge (\Delta \approx_{\mathrm{I}} n)\} \cup \{(\Omega, \Delta) | \exists n \in \mathbb{N}.\, (\Omega \approx_{\mathrm{D}} n) \wedge (\Delta \approx_{\mathrm{D}} n)\}\,.$$

We shall show that $\mathcal{R}$ progresses to its reflexive closure and then conclude, by ??, that $\mathcal{R}$ is contained within rewriting bisimilarity.

Immediate output bisimulation Assume that $\Omega\ \mathcal{R}\ \Delta = \underleftarrow{\Delta}'_L\ \Delta'\ \underrightarrow{\Delta}'_R$. According to big-step adequacy of decrements (theorem 7.23), there are three cases that derive from Δ.

- Consider the case in which $\Omega \approx_{\text{D}} 0$ and $\Delta = \underrightarrow{z} \approx_{\text{D}} 0$. According to big-step adequacy of decrements (theorem 7.23), $\Omega \Longrightarrow \underrightarrow{z}$ and so, indeed, $\Omega \Longrightarrow (\mathcal{R}^{=}\underrightarrow{z})\ \underrightarrow{z}$.
- Consider the case in which $\Omega \approx_{\text{D}} n$ and $n > 0$ and $\Delta = \Delta'\ \underrightarrow{s}$ and $\Delta' \approx_{\text{I}} n - 1$. According to big-step adequacy of decrements (theorem 7.23), $\Omega \Longrightarrow \Omega'\ \underrightarrow{s}$ for some Ω' such that $\Omega' \approx_{\text{I}} n-1$. It immediately follows that $\Omega \Longrightarrow (\mathcal{R}\underrightarrow{s})\ \Delta$.
- Consider the case in which $\Omega \approx_{\text{D}} n$ and $\Delta \approx_{\text{D}} n$, with both $\underleftarrow{\Delta}'_L$ and $\underrightarrow{\Delta}'_R$ empty. The required $\Omega \Longrightarrow (\underleftarrow{\Delta}'_L\mathcal{R}^{=}\underrightarrow{\Delta}'_R)\ \Delta$ is trivial.

Immediate input bisimulation Assume that $\Omega\ \mathcal{R}\ \Delta$ and $\underrightarrow{\Delta}_L\ [\Delta]\ \underleftarrow{\Delta}_R \longrightarrow \Delta'$. There are several cases.

- Consider the case in which $\Omega \approx_{\text{I}} n$ and $\Delta \approx_{\text{I}} n$, for some n. According to ??, the input transition is either $[\Delta]\ \underleftarrow{i} \longrightarrow \Delta'$ or $[\Delta]\ \underleftarrow{d} \longrightarrow \Delta'$.
  For the former transition, we may apply the $\underleftarrow{i}$-I rule to deduce that $\Omega\ \underleftarrow{i} \approx_{\text{I}} n+1$ and $\Delta\ \underleftarrow{i} \approx_{\text{I}} n+1$. Because $\Delta\ \underleftarrow{i} \longrightarrow \Delta'$, it follows from preservation(??) that $\Delta' \approx_{\text{I}} n+1$. We conclude that $\Omega\ \underleftarrow{i} \Longrightarrow \mathcal{R}\ \Delta'$, as required.
  For the latter input transition, a similar argument can be used.
- Consider the case in which $\Omega \approx_{\text{D}} n$ and $\Delta \approx_{\text{D}} n$, for some n. By ??, the input transition can only be $[\Delta] \longrightarrow \Delta'$, and so $\Delta \longrightarrow \Delta'$. It follows from preservation(??) that $\Delta' \approx_{\text{D}} n$; we conclude that $\Omega \Longrightarrow \mathcal{R}\ \Delta'$, as required.

Reduction bisimulation Assume that $\Omega\ \mathcal{R}\ \Delta \longrightarrow \Delta'$. The states Ω and Δ are either both increment- or both decrement-states with equal denotations. In either case, the relevant preservation result (??) allows us to deduce that Δ' has the same denotation and then conclude that $\Omega \Longrightarrow \mathcal{R}\ \Delta'$, as required.

Emptiness bisimulation This is vacuously true.

- To prove the converse, that bisimilar states have equal denotations, we shall use a lexicographic induction, first on the denotation n and then on [..].

  1. Assume that Ω and Δ are bisimilar decrement states denoting n and n', respectively. We distinguish cases on n.
     - Consider the case in which $n = 0$. By big-step adequacy of decrements (theorem 7.23), $\Omega \Longrightarrow \underrightarrow{z}$. Because Ω and Δ are bisimilar,

$\Delta \Longrightarrow(\cong \underset{\rightarrow}{z})\ \underset{\rightarrow}{z}$. According to theorem 7.23 again, 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 (theorem 7.23), $\Omega \Longrightarrow \Omega'\ \underset{\rightarrow}{s}$ 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 theorem 7.23 again, 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 Ω and Δ are bisimilar increment states denoting n and n', respectively. 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 (??), $\Omega\ \underset{\leftarrow}{d} \cong \Delta\ \underset{\leftarrow}{d}$. By part ?? of the inductive hypothesis, we conclude that $n = n'$, as required. □

7.10.2 *An alternative specification of a binary counter*

The above description of a binary counter, repeated here for convenience, could be described as object-oriented. Like objects, the processes e, b_0, and b_1 dispatch on incoming messages $\underset{\leftarrow}{i}$ and $\underset{\leftarrow}{d}$, and the process b_0' dispatches on incoming messages $\underset{\rightarrow}{z}$ and $\underset{\rightarrow}{s}$.[18]

Alternatively, we could specify the binary counter in a dual way: like functions are applied to data, the processes i and d act on incoming messages $\underset{\rightarrow}{e}$, $\underset{\rightarrow}{b}_0$, and $\underset{\rightarrow}{b}_1$, and the processes z and s act on incoming $\underset{\leftarrow}{b}_0'$ messages.

$$
\begin{aligned}
i &\triangleq (\underset{\rightarrow}{e} \setminus \underset{\rightarrow}{e} \bullet \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_0 \setminus \underset{\rightarrow}{b}_1) \,\&\, (\underset{\rightarrow}{b}_1 \setminus i \bullet \underset{\rightarrow}{b}_0) \\
d &\triangleq (\underset{\rightarrow}{e} \setminus \uparrow\downarrow z) \,\&\, (\underset{\rightarrow}{b}_0 \setminus d \bullet \underset{\leftarrow}{b}_0') \,\&\, (\underset{\rightarrow}{b}_1 \setminus \underset{\rightarrow}{b}_0 \bullet s) \\
z &\triangleq \uparrow\downarrow z \,/\, \underset{\leftarrow}{b}_0' \\
s &\triangleq \underset{\rightarrow}{b}_1 \bullet s \,/\, \underset{\leftarrow}{b}_0'
\end{aligned}
$$

In contrast with the earlier object-oriented specification, this specification could be described as functional in style.

$$\underset{\rightarrow}{e}\ \underset{\rightarrow}{b}_1\ i \Longrightarrow \underset{\rightarrow}{e}\ i\ \underset{\rightarrow}{b}_0 \Longrightarrow \underset{\rightarrow}{e}\ \underset{\rightarrow}{b}_1\ \underset{\rightarrow}{b}_1$$

Intuitively, we should expect these two specifications to be equivalent descriptions of a binary counter. To make this equivalence concrete, we might imagine defining a binary relation $\mathcal{D}$ on binary counters that makes the duality precise; for example, $e\ b_1\ \underset{\leftarrow}{i}\ \mathcal{D}\ \underset{\rightarrow}{e}\ \underset{\rightarrow}{b}_1\ i$.

However, in defining the duality relation, we implicitly observe and compare the counters' internal structures. Although certainly possible at the meta-level, this is somewhat unsatisfying because it doesn't compare the counters' *behaviors*. There ought to be a way to characterize the counters' equivalence using bisimilarity.

Doing so requires a few small changes to

$$
\begin{aligned}
e &\triangleq (e \bullet b_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\rightarrow}{z} \,/\, \underset{\leftarrow}{d}) \\
b_0 &\triangleq (\uparrow\downarrow b_1 \,/\, \underset{\leftarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet b_0' \,/\, \underset{\leftarrow}{d}) \\
b_1 &\triangleq (\underset{\leftarrow}{i} \bullet b_0 \,/\, \underset{\leftarrow}{i}) \,\&\, (b_0 \bullet \underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{d}) \\
b_0' &\triangleq (\underset{\rightarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus b_1 \bullet \underset{\rightarrow}{s})
\end{aligned}
$$

Figure 7.6: An object-oriented specification of a binary counter

[18] For a study of the relationship between (session-typed) processes and objects, see **Balzer+Pfenning:AGERE15**.

$$c \triangleq (i \bullet c \;/\; \underleftarrow{i}) \;\&\; (d \bullet \underleftarrow{u} \;/\; \underleftarrow{d})$$
$$z \triangleq (\uparrow\downarrow z \;/\; \underleftarrow{b'_0}) \;\&\; (\underrightarrow{z} \;/\; \underleftarrow{u})$$
$$s \triangleq (\underrightarrow{b_1} \bullet s \;/\; \underleftarrow{b'_0}) \;\&\; (c \bullet \underrightarrow{s} \;/\; \underleftarrow{u})$$

$$\frac{}{\underrightarrow{e} \approx^\circ_{\text{I}} 0} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, \underrightarrow{b_0} \approx^\circ_{\text{I}} 2n} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, \underrightarrow{b_1} \approx^\circ_{\text{I}} 2n+1} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, i \approx^\circ_{\text{I}} n+1}$$

$$\frac{}{\underrightarrow{e} \bullet \underrightarrow{b_1} \approx^\circ_{\text{I}} 1} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (i \bullet \underrightarrow{b_0}) \approx^\circ_{\text{I}} 2(n+1)}$$

$$\frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, c \approx_{\text{I}} n} \qquad \frac{\Delta \approx_{\text{I}} n}{\Delta\, \underleftarrow{i} \approx_{\text{I}} n+1} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (i \bullet c) \approx_{\text{I}} n+1}$$

THEOREM 7.26 (small-step adequacy of increments).
Value soundness
Preservation *If* $\Delta \approx_{\text{I}} n$ *and* $\Delta \longrightarrow \Delta'$, *then* $\Delta' \approx_{\text{I}} n$.
Progress *If* $\Delta \approx_{\text{I}} n$, *then either:* $\bullet\, \Delta \longrightarrow \Delta'$, *for some* Δ'; *or* $\bullet\, \Delta \approx_{\text{V}} n$.
Termination *If* $\Delta \approx_{\text{I}} n$, *then every rewriting sequence from* Δ *is finite.*

$$\frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, d \approx^\circ_{\text{D}} n} \qquad \frac{\Delta \approx^\circ_{\text{D}} n}{\Delta\, \underleftarrow{b'_0} \approx^\circ_{\text{D}} 2n} \qquad \frac{}{z \approx^\circ_{\text{D}} 0} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, s \approx^\circ_{\text{D}} n+1}$$

$$\frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (d \bullet \underleftarrow{b'_0}) \approx^\circ_{\text{D}} 2n} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (\underrightarrow{b_0} \bullet s) \approx^\circ_{\text{D}} 2n+1} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (\underrightarrow{b_1} \bullet s) \approx^\circ_{\text{D}} 2n+2}$$

$$\frac{\Delta \approx_{\text{I}} n}{\Delta\, \underleftarrow{d} \approx_{\text{D}} n} \qquad \frac{\Delta \approx^\circ_{\text{D}} n}{\Delta\, \underleftarrow{u} \approx_{\text{D}} n} \qquad \frac{}{\underrightarrow{z} \approx_{\text{D}} 0} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, c\, \underrightarrow{s} \approx_{\text{D}} n+1}$$

$$\frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (d \bullet \underleftarrow{u}) \approx_{\text{D}} n} \qquad \frac{\Delta \approx^\circ_{\text{I}} n}{\Delta\, (c \bullet \underrightarrow{s}) \approx_{\text{D}} n+1}$$

THEOREM 7.27 (Small-step adequacy of decrements).
Preservation *If* $\Delta \approx_{\text{D}} n$ *and* $\Delta \longrightarrow \Delta'$, *then* $\Delta' \approx_{\text{D}} n$.
Progress *If* $\Delta \approx_{\text{D}} n$, *then either:*

- $\Delta \longrightarrow \Delta'$ *for some* Δ';
- $n = 0$ *and* $\Delta = \underrightarrow{z}$;
- $n > 0$ *and* $\Delta = \Delta'\, c\, \underrightarrow{s}$, *for some* Δ' *such that* $\Delta' \approx^\circ_{\text{I}} n-1$.

Termination *If* $\Delta \approx_{\text{D}} n$, *then every rewriting sequence from* Δ *is finite.*

THEOREM 7.28 (Big-step adequacy of decrements). *If* $\Delta \approx_{\text{D}} n$, *then:*

- $\Delta \Longrightarrow \underleftarrow{\Delta'_L}\; \Delta'\; \underrightarrow{\Delta'_R}$ *only if either:* $\underleftarrow{\Delta'_L} = \underrightarrow{\Delta'_R} = \cdot$; *or* $n = 0$ *and* $\underleftarrow{\Delta'_L} = \cdot$ *and* $\underrightarrow{\Delta'_R} = \underrightarrow{z}$; *or* $n > 0$ *and* $\underleftarrow{\Delta'_L} = \cdot$ *and* $\underrightarrow{\Delta'_R} = \underrightarrow{s}$;
- $\Delta \Longrightarrow \underrightarrow{z}$ *if* $n = 0$;

- $\Delta \Longrightarrow \Delta' \; c \; \underset{\rightarrow}{s}$ *for some* Δ' *such that* $\Delta' \approx_{\mathrm{I}}^{\circ} n - 1$, *if* $n > 0$; *and*
- $\Delta \Longrightarrow \Delta' \; \underset{\rightarrow}{s}$ *only if* $n > 0$ *and* $\Delta' = \Delta' \; c$ *for some* Δ' *such that* $\Delta' \approx_{\mathrm{I}}^{\circ} n - 1$.

THEOREM 7.29 (Big-step adequacy of decrements). *If* $\Delta \approx_{\mathrm{D}} n$, *then:*

- $\Delta \Longrightarrow \underset{\rightarrow}{z}$ *if, and only if,* $n = 0$;
- $\Delta \Longrightarrow \Delta' \; c \; \underset{\rightarrow}{s}$ *for some* Δ' *such that* $\Delta' \approx_{\mathrm{I}}^{\circ} n - 1$, *if* $n > 0$; *and*
- $\Delta \Longrightarrow \Delta' \; \underset{\rightarrow}{s}$ *only if* $n > 0$ *and* $\Delta' = \Delta' \; c$ *for some* Δ' *such that* $\Delta' \approx_{\mathrm{I}}^{\circ} n - 1$.

THEOREM 7.30. *If* $\Omega \approx_{\mathrm{I}} n$ *and* $\Delta \approx_{\mathrm{I}} n'$, *then* $\Omega \cong \Delta$ *if, and only if,* $n = n'$. *Similarly, if* $\Omega \approx_{\mathrm{D}} n$ *and* $\Delta \approx_{\mathrm{D}} n'$, *then* $\Omega \cong \Delta$ *if, and only if,* $n = n'$.

Proof.

- $\Omega \approx_{\mathrm{D}} 0$ and $\underset{\rightarrow}{z} \approx_{\mathrm{D}} 0$. $\Omega \Longrightarrow \underset{\rightarrow}{z}$
- $\Omega \approx_{\mathrm{D}} n + 1$ and $\Delta \; c \; \underset{\rightarrow}{s} \approx_{\mathrm{D}} n + 1$. $\Omega \Longrightarrow \Omega' \; \underset{\rightarrow}{s}$ and $\Omega' \approx_{\mathrm{I}} n$. $\Omega' \; \mathcal{R} \; \Delta \; c$

□

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.

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 **??**. [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.]

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

[1] Or, at most one consequent if multiplicative falsehood is included(see, for example, **??**).

[2] **Gentzen**:.

[3] **??**.

[4] Steinberger 2011.

[5] **Harper**:**??**.

metatheory. Together, the cut elimination and identity elimination metatheorems identify the cut-free, η-long proofs as verifications that [form the foundation] exhibit a subformula property.

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. An analysis of the its basic metatheory(**??**) begins to suggest the basis of a Curry–Howard interpretation of Hilbert-style proofs as chains of well-typed, asynchronously communicating processes. **??** will be devoted to developing that observation more fully.

Finally, **??** 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[6] – proof theory, category theory, and type theory:

[6] **Harper:??**.

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, so singleton contexts form only the degenerate algebraic structure of a set.

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

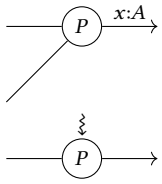
monoidal product:

$$\ulcorner \mathcal{D} :: (A_1, A_2, \ldots, A_n \vdash B) \urcorner : \ulcorner A_1 \urcorner \otimes \ulcorner A_2 \urcorner \otimes \cdots \otimes \ulcorner A_n \urcorner \to \ulcorner B \urcorner$$

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 **??**'s SILL type theory 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 linear 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.



8.2 *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 **??**, 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 sequents [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 **??**). Rules for the other additive connectives ($\&$, $\top$, and $\mathbf{0}$) can be constructed, too, but we will momentarily postpone displaying them.

$$\dfrac{\Omega \vdash B \quad \Omega'_L\, B\, \Omega'_R \vdash C}{\Omega'_L\, \Omega\, \Omega'_R \vdash C}\ \textsc{cut}^B$$

$$\vdots$$

$$\dfrac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B$$

Figure 8.1: Deriving the singleton sequent calculus's cut rule from the corresponding ordered sequent calculus rule

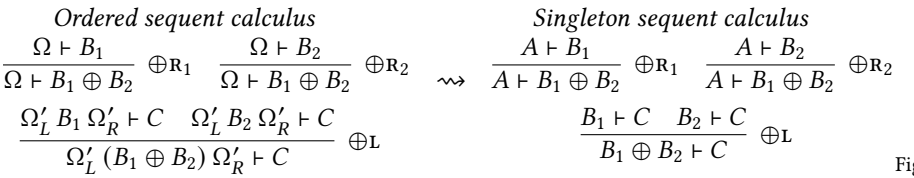


Figure 8.2: Deriving the singleton sequent calculus rules for fig:singleton-logic:seq-calc:derive-cut cut and fig:singleton-logic:seq-calc:derive-plus additive disjunction from the corresponding ordered sequent calculus rules

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. [, precisely because their multiplicative nature involves splitting antecedents among several premises and, in other rules, extending the context with additional antecedents.]

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.

In trying to construct singleton sequent calculus rules for left-handed implication, the fundamental problem is that the $\backslash\mathrm{R}$ 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.

$$\frac{B_1\ \Omega \vdash B_2}{\Omega \vdash B_1 \setminus B_2}\ \backslash\mathrm{R}$$
$$\vdots$$
$$\frac{B_1\ A \vdash B_2}{A \vdash B_1 \setminus B_2}\ \backslash\mathrm{R}?$$
$$\vdots$$
$$\frac{B_1 \bullet A \vdash B_2}{A \vdash B_1 \setminus B_2}\ \backslash\mathrm{R}?$$

Figure 8.3: A failed attempt at constructing a right rule for left-handed implication

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 $\Omega = A \vdash B$ in the $\oplus$,$\&$-fragment of the singleton sequent calculus.*

$$\textsc{propositions} \quad A, B, C ::= \alpha \mid A \oplus B \mid \mathbf{0} \mid A \mathbin{\&} B \mid \top$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{A \vdash B_1}{A \vdash B_1 \oplus B_2}\ \oplus\textsc{r}_1 \qquad \frac{A \vdash B_2}{A \vdash B_1 \oplus B_2}\ \oplus\textsc{r}_2 \qquad \frac{B_1 \vdash C \quad B_2 \vdash C}{B_1 \oplus B_2 \vdash C}\ \oplus\textsc{l}$$

$$\text{(no } \mathbf{0}\textsc{r} \text{ rule)} \qquad \frac{}{\mathbf{0} \vdash C}\ \mathbf{0}\textsc{l}$$

$$\frac{A \vdash B_1 \quad A \vdash B_2}{A \vdash B_1 \mathbin{\&} B_2}\ \&\textsc{r} \qquad \frac{B_1 \vdash C}{B_1 \mathbin{\&} B_2 \vdash C}\ \&\textsc{l}_1 \qquad \frac{B_2 \vdash C}{B_1 \mathbin{\&} B_2 \vdash C}\ \&\textsc{l}_2$$

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

Figure 8.4: A sequent calculus for propositional singleton logic

[Should I mention problems with $\top$ and $\mathbf{0}$ and how singleton logic sanitizes them?]

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, 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,[7] 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].[8]

A *verification*, then, is a proof that relies only on the right and left inference rules and the $\textsc{id}^\alpha$ rule for propositional variables α – 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.

In the tradition of **Gentzen:??**, **Dummett:??**, and **Martin-Lof:??**,[9] a sequent calculus is well-defined if it rests on the solid foundation of a verificationist meaning-theory. 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]

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

[8] Right rules only, because it is verificationist?

[9] **Gentzen:??Dummett:??Martin-Lof:??.**

[10] Right rules only, because it is verificationist?

In **Martin-Lof:??**'s words, the meaning of a logical connective must be given by what counts as a verification of it. A *verification*, then, is a proof that relies only on the right and left inference rules and the $\textsc{id}^\alpha$ rule for propositional variables α – stated differently, verifications may not contain instances of the CUT or general $\textsc{id}^A$ rules.

For this program to succeed, we need to be sure that for every proof there is a corresponding verification – we need a weak

In this sense, the usual cut elimination metatheorem states a weak normalization result.

THEOREM 8.2 (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 (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 **??**'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 **??** 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: $A \vdash^{\text{cf}} C$ is a cut-free proof of $A \vdash C$. The admissible cut principle stated in **??** could then be expressed as the rule

$$\frac{A \vdash^{\text{cf}} B \qquad B \vdash^{\text{cf}} C}{A \vdash^{\text{cf}} C}\ \textsc{a-cut}^B,$$

with the dotted line indicating that it is an admissible, not primitive, rule. Writing it in this way emphasizes that proving **??** 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 $B \vdash C$.

$$\frac{A \vdash B \qquad B \vdash C}{A \vdash C}\ \textsc{cut}^B$$

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 the admissible A-CUT rule may only be applied to cut-free proofs.

[11] Contrast this with a separate, intrinsically cut-free sequent calculus in the style of Church (**Pfenning:Andrews??**).

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:

$$\dfrac{\dfrac{\mathcal{D}_1}{A \vdash B_1}\,\oplus\text{R}_1 \qquad \dfrac{\dfrac{\mathcal{E}_1}{B_1 \vdash C} \quad \dfrac{\mathcal{E}_2}{B_2 \vdash C}}{B_1 \oplus B_2 \vdash C}\,\oplus\text{L}}{A \vdash C}\,\text{A-CUT}^{B_1 \oplus B_2} \quad = \quad \dfrac{\dfrac{\mathcal{D}_1}{A \vdash B_1} \quad \dfrac{\mathcal{E}_1}{B_1 \vdash C}}{A \vdash C}\,\text{A-CUT}^{B_1}$$

Notice that the interaction between proofs here is synchronous – the case is resolved by appealing to the inductive hypothesis at a smaller cut formula.

Identity cases In the identity cases, one of the proofs is the ID rule alone. For example:

$$\dfrac{\dfrac{}{A \vdash A}\,\text{ID}^A \qquad \dfrac{\mathcal{E}}{A \vdash C}}{A \vdash C}\,\text{A-CUT}^A \quad = \quad \dfrac{\mathcal{E}}{A \vdash C}\,.$$

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:

$$\dfrac{\dfrac{\mathcal{D}}{A \vdash B} \qquad \dfrac{\dfrac{\mathcal{E}_1}{B \vdash C_1}}{B \vdash C_1 \oplus C_2}\,\oplus\text{R}_1}{A \vdash C_1 \oplus C_2}\,\text{A-CUT}^B \quad = \quad \dfrac{\dfrac{\mathcal{D}}{A \vdash B} \quad \dfrac{\mathcal{E}_1}{B \vdash C_1}}{A \vdash C_1}\,\text{A-CUT}^B \qquad \dfrac{}{A \vdash C_1 \oplus C_2}\,\oplus\text{R}_1$$

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 a cut-free proof of $A \vdash C$ exists.*

Proof. By structural induction on the proof of $A \vdash C$, appealing to the admissibility of cut(**??**) 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.

$$\dfrac{\dfrac{\mathcal{D}_1 \quad \mathcal{D}_2}{A \vdash B \quad B \vdash C}\,\textsc{cut}^B}{\dfrac{A \vdash C}{A \vdash^{\text{cf}} C}\,\textsc{ce}} \quad = \quad \dfrac{\dfrac{\mathcal{D}_1}{A \vdash B}\,\textsc{ce} \qquad \dfrac{\mathcal{D}_2}{B \vdash C}\,\textsc{ce}}{A \vdash^{\text{cf}} C}\,\textsc{a-cut}^B$$

All other cases are handled compositionally.

This proof amounts to defining a meta-level function for normalizing proofs to cut-free form. □

8.3 *A Hilbert-style axiomatization of 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. Our interest in a Hilbert system for singleton logic is not taxonomic, however. Rather, over the course of the next chapter and a half, we shall see that normalization of Hilbert-style proofs serves as the basis of a Curry–Howard isomorphism with chains of asynchronously communicating processes.

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.

$$\dfrac{\vdash A \supset B \quad \vdash A}{\vdash B}\,\textsc{mp}$$

Figure 8.5: *Modus ponens* for a Hilbert-style axiomatization of intuitionistic logic

However, as explained in **??**, singleton logic does not enjoy the luxury of an implication connective. So in a Hilbert-style axiomatization of singleton logic, we will have to content ourselves with a broad interpretation of the term 'axiom schema' that encompasses zero-premise rules.

TO CONSTRUCT a Hilbert-style axiomatization of singleton logic, we will ask, in turn, whether each sequent calculus rule can be reduced to an axiom 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 Hilbert system.

$$\dfrac{}{A \vdash A}\,\textsc{id}^A$$

$$\dfrac{A \vdash B \quad B \vdash C}{A \vdash C}\,\textsc{cut}^B$$

The CUT rule is not quite 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 [in our Hilbert system]. 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 Hilbert system.

Next, consider the sequent calculus's $\oplus\text{R}_1$ inference rule.[12] Using the ID axiom schema, we can obtain a zero-premise derived rule from $\oplus\text{R}_1$:

[12] Citation of basic logic (Sambin et al. 2000)?

$$\dfrac{\dfrac{}{A_1 \vdash A_1}\,\text{ID}}{A_1 \vdash A_1 \oplus A_2}\,\oplus\text{R}_1 \quad\leftrightsquigarrow\quad \dfrac{}{A_1 \vdash A_1 \oplus A_2}\,\oplus\text{R}'_1$$

Moreover, by combining this new $\oplus\text{R}'_1$ axiom schema with CUT, we can recover the original $\oplus\text{R}_1$ rule as a derived rule:

$$\dfrac{A \vdash B_1 \quad \dfrac{}{B_1 \vdash B_1 \oplus B_2}\,\oplus\text{R}'_1}{A \vdash B_1 \oplus B_2}\,\text{CUT} \quad\leftrightsquigarrow\quad \dfrac{A \vdash B_1}{A \vdash B_1 \oplus B_2}\,\oplus\text{R}_1$$

Together, these two observations suggest that $\oplus\text{R}'_1$ be adopted as an axiom schema in the Hilbert-style axiomatization of singleton logic. A symmetric $\oplus\text{R}'_2$ axiom schema should be adopted, too.

What about the sequent calculus's $\oplus\text{L}$ 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 $\oplus\text{L}$ rule's hypothetical judgments to categorical judgments. Like CUT, the sequent calculus's $\oplus\text{L}$ rule is thus adopted wholesale in singleton logic's Hilbert system. Including the additive $\oplus\text{L}$ rule as a primitive rule of inference is perhaps not unexpected. It is consistent with Hilbert-style axiomatizations of linear logic,[13] which include an adjunction rule – essentially the linear sequent calculus's $\&\text{R}$ rule – to effect the additive behavior that linear implication and its multiplicative *modus ponens* rule cannot.

$$\dfrac{B_1 \vdash C \quad B_2 \vdash C}{B_1 \oplus B_2 \vdash C}\,\oplus\text{L}$$

[13] Avron 1988.

The axiomatization of additive conjunction is dual to that of $A_1 \oplus A_2$: The sequent calculus's $\&\text{R}$ rule will be adopted wholesale, and $\&\text{L}'_1$ and $\&\text{L}'_2$ axiom schemas will be derived from the sequent calculus's $\&\text{L}_1$, $\&\text{L}_2$, and ID rules. And finally, the axiomatizations of $\mathbf{0}$ and $\top$ are the nullary analogues of those of the binary $\oplus$ and $\&$ connectives, respectively.

Figure 8.7 summarizes this Hilbert-style axiomatization of singleton logic.

$$\dfrac{A \vdash C_1 \quad A \vdash C_2}{A \vdash C_1 \,\&\, C_2}\,\&\text{R}$$

$$\dfrac{}{C_1 \,\&\, C_2 \vdash C_1}\,\&\text{L}'_1 \qquad \dfrac{}{C_1 \,\&\, C_2 \vdash C_2}\,\&\text{L}'_2$$

Figure 8.6: A Hilbert-style axiomatization of additive conjunction from singleton logic

THE HILBERT SYSTEM of fig. 8.7 shares so many rules with the singleton sequent calculus(fig. 8.4) that it can be viewed as a variant in which each connective's non-invertible rules have been replaced with zero-premise rules. As such, we should seek to prove the usual sequent calculus metatheorems – cut elimination and identity expansion – for this Hilbert-style variant.

Strictly speaking, cut elimination does not hold for the Hilbert system. As a concrete [counter]example, there is no cut-free Hilbert-style proof of $\alpha_2 \vdash$

$$\textsc{propositions} \quad A ::= \alpha \mid A \oplus B \mid \mathbf{0} \mid A \,\&\, B \mid \top$$

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{cut}^B \qquad \frac{}{A \vdash A}\ \textsc{id}^A$$

$$\frac{}{A_1 \vdash A_1 \oplus A_2}\ \oplus\textsc{r}'_1 \qquad \frac{}{A_2 \vdash A_1 \oplus A_2}\ \oplus\textsc{r}'_2 \qquad \frac{A_1 \vdash C \quad A_2 \vdash C}{A_1 \oplus A_2 \vdash C}\ \oplus\textsc{l}$$

$$\text{(no } \mathbf{0}\textsc{r} \text{ rule)} \qquad \frac{}{\mathbf{0} \vdash C}\ \mathbf{0}\textsc{l}$$

$$\frac{A \vdash C_1 \quad A \vdash C_2}{A \vdash C_1 \,\&\, C_2}\ \&\textsc{r} \qquad \frac{}{C_1 \,\&\, C_2 \vdash C_1}\ \&\textsc{l}'_1 \qquad \frac{}{C_1 \,\&\, C_2 \vdash C_2}\ \&\textsc{l}'_2$$

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

Figure 8.7: A Hilbert system for singleton logic

$\alpha_1 \oplus (\alpha_2 \oplus \alpha_3)$, even though the same sequent is provable using cut:

$$\dfrac{\dfrac{}{\alpha \vdash \top}\ \top\textsc{r} \quad \dfrac{}{\top \vdash \top \oplus \top}\ \oplus\textsc{r}'_1}{\alpha \vdash \top \oplus \top}\ \textsc{cut}$$

$$\dfrac{\dfrac{}{\alpha_2 \vdash \alpha_2 \oplus \alpha_3}\ \oplus\textsc{r}'_1 \quad \dfrac{}{\alpha_2 \oplus \alpha_3 \vdash \alpha_1 \oplus (\alpha_2 \oplus \alpha_3)}\ \oplus\textsc{r}'_2}{\alpha_2 \vdash \alpha_1 \oplus (\alpha_2 \oplus \alpha_3)}\ \textsc{cut}$$

Although cut elimination does not hold, normal forms nevertheless exist. Normal Hilbert-style 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 Hilbert system*

Before presenting a proof of non-analytic cut elimination, we will take a moment to introduce a proof term assignment for the Hilbert system. 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 8.8 presents the labeled, n-ary generalization of the singleton Hilbert system, 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 compostion, $f_2 \circ f_1$.[14] The proof term $\leftrightarrow$ is used for the ID rule. Because of their sim-

[14] 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.

$$\textsc{propositions}\quad A ::= \alpha \mid \oplus_{\ell\in L}\{\ell : A_\ell\} \mid \&_{\ell\in L}\{\ell : A_\ell\}$$

$$\textsc{proof terms}\quad P ::= P_1 \diamond P_2 \mid \leftrightarrow \mid \underrightarrow{k} \mid \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow P_\ell) \mid \mathsf{caseR}_{\ell\in L}(\ell \Rightarrow P_\ell) \mid \underleftarrow{k}$$

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \leftrightarrow : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell\in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon\ A_\ell \vdash P_\ell : C}{\oplus_{\ell\in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon\ A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell\in L}(\ell \Rightarrow P_\ell) : \&_{\ell\in L}\{\ell : C_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell\in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\textsc{l}'$$

Figure 8.8: Proof terms for a labeled, n-ary variant of the Hilbert system of fig. 8.7

ilar structure, the $\oplus\textsc{r}'$ and $\&\textsc{l}'$ rules are assigned the similar proof terms $\underrightarrow{k}$ and $\underleftarrow{k}$; the direction of the underlying arrow distinguishes them. Similarly, the $\oplus\textsc{l}$ and $\&\textsc{r}$ rules are assigned the proof terms $\mathsf{caseL}_{\ell\in L}(\ell \Rightarrow P_\ell)$ and $\mathsf{caseR}_{\ell\in L}(\ell \Rightarrow P_\ell)$.

- Variable-free combinators

8.3.2 *Non-analytic cut elimination for the singleton Hilbert system*

With proof terms in hand, we can now return to our goal of establishing a *non-analytic* cut elimination **??** for the singleton Hilbert system.

The cut elimination procedure will normalize a Hilbert-style proof so that any remaining cuts are analytic, specifically of the forms $\underleftarrow{k} \diamond P$ or $P \diamond \underrightarrow{k}$. As shown in the neighboring display, cuts of these forms are analytic because the cut formula is a subformula of the conclusion sequent.

$$\frac{\dfrac{(k \in L)}{\&_{\ell\in L}\{\ell : A_\ell\} \vdash \underleftarrow{k} : A_k}\ \&\textsc{l}' \qquad A_k \vdash P : C}{\&_{\ell\in L}\{\ell : A_\ell\} \vdash \underleftarrow{k} \diamond P : C}\ \textsc{cut}^{A_k}$$

$$\frac{A \vdash P : C_k \qquad \dfrac{(k \in L)}{C_k \vdash \underrightarrow{k} : \oplus_{\ell\in L}\{\ell : C_\ell\}}\ \oplus\textsc{r}'}{A \vdash P \diamond \underrightarrow{k} : \oplus_{\ell\in L}\{\ell : C_\ell\}}\ \textsc{cut}^{C_k}$$

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.

$$\textsc{normal terms}\quad N, M ::= \leftrightarrow \mid N \diamond \underrightarrow{k} \mid \underleftarrow{k} \diamond N \mid \underrightarrow{k} \mid \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow N_\ell) \mid \mathsf{caseR}_{\ell\in L}(\ell \Rightarrow N_\ell) \mid \underleftarrow{k}$$

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 (Non-analytic cut elimination). *If $A \vdash P : C$, then $A \vdash N : C$ for some normal term N.*

As for the [singleton] sequent calculus's cut elimination result(theorem 8.2), this **??** can be proved by a straightforward structural induction, this time on the given term, P. First, however, we need admissibility of non-analytic cut as a lemma:

$$\dfrac{A \vdash N_0 : B_k \quad \dfrac{(k \in L)}{B_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\oplus\text{R}'}{A \vdash N_0 \diamond \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\text{CUT}^{B_k} \qquad \oplus_{\ell \in L}\{\ell : B_\ell\} \vdash M : C}{A \vdash (N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M : C}\,\text{A-CUT}^{B}$$

$$=$$

$$\dfrac{A \vdash N_0 : B_k \quad \dfrac{\dfrac{(k \in L)}{B_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : B_\ell\}}\,\oplus\text{R}' \qquad \oplus_{\ell \in L}\{\ell : B_\ell\} \vdash M : C}{B_k \vdash \underset{\rightarrow}{k} \blacklozenge M : C}\,\text{A-CUT}^{B}}{A \vdash N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M) : C}\,\text{A-CUT}^{B_k}$$

Figure 8.9: One of the associative cases in the proof of non-analytic cut admissibility(lemma 8.5)

Lemma 8.5 (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 **??** 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.

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 Hilbert system 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 become smaller – $\underset{\rightarrow}{k}$ is a proper subterm of $N_0 \diamond \underset{\rightarrow}{k}$ – while the cut formula and other term remain 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.[15]

The symmetric case, $N \blacklozenge (\underset{\leftarrow}{k} \diamond M_0)$, is also an associative case and is

[15] To aid the reader in tracking the types, fig. 8.9 shows the full typing derivations.

handled similarly. The complete set of associative clauses is therefore:

$$(N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M = N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)$$
$$N \blacklozenge (\underset{\leftarrow}{k} \diamond M_0) = (N \blacklozenge \underset{\leftarrow}{k}) \blacklozenge M_0$$

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 $N_0 \diamond \underset{\rightarrow}{k}$ and $\underset{\leftarrow}{k} \diamond M_0$, 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:

$$\underset{\rightarrow}{k} \blacklozenge \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$$
$$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge \underset{\leftarrow}{k} = N_k$$

If $\underset{\rightarrow}{k}$ and $\underset{\leftarrow}{k}$ 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 the singleton Hilbert system that we develop in the following chapter.

Identity cases As in the proof of admissibility of cut for the sequent calculus(**??**), 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 $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge M$, 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 $\blacklozenge$.

$$(\underset{\leftarrow}{k} \diamond N_0) \blacklozenge M = \underset{\leftarrow}{k} \diamond (N_0 \blacklozenge M)$$
$$\underset{\leftarrow}{k} \blacklozenge M = \underset{\leftarrow}{k} \diamond M$$
$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge M = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell \blacklozenge M)$$

In these clauses, the $\blacklozenge$ is permuted with a normal term's top-lovel constructor.

There are also several right-commutative cases that are symmetric to the preceding left-commutative cases:

$$N \blacklozenge (M_0 \diamond \underset{\rightarrow}{k}) = (N \blacklozenge M_0) \diamond \underset{\rightarrow}{k}$$
$$N \blacklozenge \underset{\rightarrow}{k} = N \diamond \underset{\rightarrow}{k}$$
$$N \blacklozenge \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow M_\ell) = \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N \blacklozenge M_\ell)$$

□

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

$$\begin{aligned}&\mathsf{caseL}_{\ell\in L}(\ell \Rightarrow N_\ell) \blacklozenge \mathsf{caseR}_{k\in K}(k \Rightarrow M_k)\\ &\quad = \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow \mathsf{caseR}_{k\in K}(k \Rightarrow N_\ell \blacklozenge M_k))\end{aligned}$$

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 two above terms, seem rather arbitrary, so we prefer to have $\blacklozenge$ remain nondeterministic.

With this ?? in hand, we may finally proceed to proving non-analytic cut elimination.

Theorem 8.6 (Non-analytic cut elimination). *If $A \vdash P : C$, then $A \vdash N : C$ for some normal term N.*

Proof.

$$\dfrac{A \vdash P : C}{A \vdash ce(P) : C}\ \textsc{ce}$$

$$\dfrac{\dfrac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B}{A \vdash ce(P_1 \diamond P_2) : C}\ \textsc{ce} \quad = \quad \dfrac{\dfrac{A \vdash P_1 : B}{A \vdash ce(P_1) : C}\ \textsc{ce} \quad \dfrac{B \vdash P_2 : C}{B \vdash ce(P_2) : C}\ \textsc{ce}}{A \vdash ce(P_1) \blacklozenge ce(P_2) : C}\ \textsc{a-cut}^B$$

$$\begin{aligned}&ce(P_1 \diamond P_2) = ce(P_1) \blacklozenge ce(P_2)\\ &ce(\leftrightarrow) = \leftrightarrow\\ &ce(\underrightarrow{k}) = \underrightarrow{k}\\ &ce(\mathsf{caseL}_{\ell\in L}(\ell \Rightarrow P_\ell)) = \mathsf{caseL}_{\ell\in L}(\ell \Rightarrow ce(P_\ell))\\ &ce(\mathsf{caseR}_{\ell\in L}(\ell \Rightarrow P_\ell)) = \mathsf{caseR}_{\ell\in L}(\ell \Rightarrow ce(P_\ell))\\ &ce(\underleftarrow{k}) = \underleftarrow{k}\end{aligned}$$

□

We posit that normalization is unique up to commuting conversions and identity reductions.

8.4

$$\eta(\alpha) = \leftrightarrow\!\!\triangleright$$
$$\eta(\oplus_{\ell \in L}\{\ell : A_\ell\}) = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow \underrightarrow{\ell})$$
$$\eta(\&_{\ell \in L}\{\ell : A_\ell\}) = \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow \underleftarrow{\ell})$$

$$\eta(\alpha) = \triangleleft\!\!\leftrightarrow$$
$$\eta(\oplus_{\ell \in L}\{\ell : A_\ell\}) = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow \eta(A_\ell) \diamond \underrightarrow{\ell})$$
$$\eta(\&_{\ell \in L}\{\ell : A_\ell\}) = \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow \underleftarrow{\ell} \diamond \eta(A_\ell))$$

$$n(A, P \diamond^B Q, C) = n(A, P, B) \blacklozenge n(B, Q, C)$$
$$n(A, \leftrightarrow, A) = \eta(A)$$
$$n(\oplus_{\ell \in L}\{\ell : A_\ell\}, \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell), C) = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow n(A_\ell, P_\ell, C))$$

8.5 *Extensions of singleton logic*

The singleton sequent calculus and singleton Hilbert system support various extensions. One simple but useful extension is to introduce universal and existential quantifiers, $\forall x{:}\tau.A$ and $\exists x{:}\tau.A$, over well-sorted data. Variable typing assumptions, $x{:}\tau$, are not 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. So now, instead of sequents $A \vdash C$, we have sequents $\delta \vdash \gamma$, where δ and γ adhere to the following grammars.

$$\text{SUBSINGLETON ANTECEDENTS} \quad \delta ::= A \mid \cdot$$
$$\text{SUBSINGLETON CONSEQUENTS} \quad \gamma ::= C \mid \cdot$$

With this relaxation, we arrive at *sub*singleton logic.

The multiplicative units $\mathbf{1}$ and $\bot$ can now be characterized by right and left rules:

$$\frac{}{\cdot \vdash \mathbf{1}}\ \mathbf{1}\mathrm{R} \qquad \frac{\cdot \vdash \gamma}{\mathbf{1} \vdash \gamma}\ \mathbf{1}\mathrm{L}$$

$$\frac{\delta \vdash \cdot}{\delta \vdash \bot}\ \bot\mathrm{R} \qquad \frac{}{\bot \vdash \cdot}\ \bot\mathrm{L}$$

These rules apply to both the sequent calculus and Hilbert system presentations of subsingleton logic.

Without the binary multiplicative connectives $\otimes$ and ...

$$\text{PROPOSITIONS} \quad A ::= \cdots \mid\ !A$$

$$\text{PERSISTENT CONTEXTS} \quad \Gamma ::= \cdot \mid \Gamma, A$$

$$\frac{\Gamma; \cdot \vdash A \quad \Gamma, A; \delta \vdash \gamma}{\Gamma, A; \delta \vdash \gamma}\ \text{CUT}!^A \qquad \frac{\Gamma, A; A \vdash \gamma}{\Gamma, A; \cdot \vdash \gamma}\ \text{COPY}$$

$$\frac{\Gamma; \cdot \vdash A}{\Gamma; \cdot \vdash\ !A}\ !\text{R} \qquad \frac{\Gamma, A; \cdot \vdash \gamma}{\Gamma;\ !A \vdash \gamma}\ !\text{L}$$

$$\frac{\Gamma; \cdot \vdash A_1 \quad \Gamma; \delta \vdash A_2}{\Gamma; \delta \vdash A_1\ !\otimes\ A_2}\ !\otimes\text{R} \qquad \frac{\Gamma, A_1; A_2 \vdash \gamma}{\Gamma; A_1\ !\otimes\ A_2 \vdash \gamma}\ !\otimes\text{L}$$

$$\frac{}{\Gamma, A; \cdot \vdash\ !A}\ !\text{R} \qquad \frac{}{\Gamma, A_1; A_2 \vdash A_1\ !\otimes\ A_2}\ !\otimes\text{R}$$

8.6 *Related work*

- Fortier and Santocanale paper using (synchronous) singleton logic
- CSL '12 paper on asynchronous Curry–Howard for linear logic

8.7 *Hilbert*

8.7.1 *Hypothetical Hilbert system*

$$\frac{\Gamma \vdash A \to B\ \mathit{hil} \quad \Gamma \vdash A\ \mathit{hil}}{\Gamma \vdash B\ \mathit{hil}}\ \text{MP} \qquad \frac{}{\Gamma, A\ \mathit{hil} \vdash A\ \mathit{hil}}\ \text{HYP}$$

$$\Gamma \vdash A \to A\ \mathit{hil}$$
$$\Gamma \vdash A \to (B \to A)\ \mathit{hil}$$
$$\Gamma \vdash (A \to B \to C) \to (A \to B) \to (A \to C)\ \mathit{hil}$$

THEOREM 8.7. *If* $\Gamma \vdash A$, *then* $\hat{\Gamma} \vdash A\ \mathit{hil}$.

Proof.

$$\frac{\Gamma, A \vdash B}{\Gamma \vdash A \to B}\ \to\text{R}$$

We need a deduction theorem.

$$\frac{\Gamma, A \to B \vdash A \quad \Gamma, A \to B, B \vdash C}{\Gamma, A \to B \vdash C}\ \to\text{L}$$

We have $\Gamma, A \to B \vdash A\ \mathit{hil}$ by induction. We also have $\Gamma, A \to B \vdash B \to C\ \mathit{hil}$ by induction and the deduction theorem. Prove $\Gamma, A \to B \vdash B\ \mathit{hil}$ by HYP and MP. Conclude $\Gamma, A \to B \vdash C\ \mathit{hil}$ by MP.

$$\frac{\Gamma \vdash A \quad \Gamma, A \vdash C}{\Gamma \vdash C}\ \textsc{cut}$$

Apply the inductive hypothesis and deduction theorem, then MP. □

8.7.2

$$[\![x]\!]\,\rho = \rho(x)$$
$$[\![\lambda x.M]\!]\,\rho\,v = [\![M]\!]\,(\rho[x \mapsto v])$$
$$[\![M\,N]\!]\,\rho = [\![M]\!]\,\rho\,([\![N]\!]\,\rho)$$

$$[\![0]\!]\,(\rho, v_0) = v_0 \qquad [\![n+1]\!]\,(\rho, v_0) = [\![n]\!]\,\rho$$
$$[\![\lambda x.M]\!]\,\rho\,v_0 = [\![M]\!]\,(\rho, v_0)$$
$$[\![M\,N]\!]\,\rho = [\![M]\!]\,\rho\,([\![N]\!]\,\rho)$$

$$[\![n]\!] = \pi\,n$$
$$[\![\lambda x.M]\!] = \Lambda\,[\![M]\!]$$
$$[\![M\,N]\!] = S\,[\![M]\!]\,[\![N]\!]$$

$$\pi\,0\,(x, y) = y \qquad \pi\,(n+1)\,(x, y) = \pi\,n\,x$$
$$\Lambda\,f\,x\,y = f\,(x, y)$$
$$S\,x\,y\,z = x\,z\,(y\,z)$$

$$\vdash \circ : (B \to C) \to (A \to B) \to (A \to C)$$
$$\vdash \iota_1 : A \to A \vee B$$
$$\vdash \iota_2 : B \to A \vee B$$
$$f{:}A \to C, g{:}B \to C \vdash [f, g] : A \vee B \to C$$

$$[f_1, f_2]\,(\iota_i\,x) = f_i\,x$$
$$[f_1, f_2] \circ \iota_i = f_i$$

$$[\![\mathsf{selectR}\ i_1; P]\!] = S_1\,[\![P]\!]$$
$$[\![\mathsf{caseL}(i_1 \Rightarrow Q_1 \mid i_2 \Rightarrow Q_2)]\!] = [[\![Q_1]\!], [\![Q_2]\!]]$$
$$[\![P \diamond Q]\!] = [\![Q]\!] \circ [\![P]\!]$$

$$[g_1, g_2](S_1\,f\,x) = g_1(f\,x) \mathit{and} S_1\,f\,x = \iota_1\,(f\,x)$$

8.8 *Natural deduction*

$$\frac{}{A \vdash A}\ \textsc{hyp} \qquad \frac{A \vdash B \quad B \vdash C}{A \vdash C}\ \textsc{subst}$$

$$\frac{A \vdash B_k \quad (k \in L)}{A \vdash \oplus_{\ell \in L}\{\ell : B_\ell\}}\ \oplus\textsc{i} \qquad \frac{A \vdash \oplus_{\ell \in L}\{\ell : B_\ell\} \quad \forall \ell \in L : B_\ell \vdash C}{A \vdash C}\ \oplus\textsc{e}$$

$$\frac{\forall \ell \in L : A \vdash B_\ell}{A \vdash \&_{\ell \in L}\{\ell : B_\ell\}}\ \&\textsc{i} \qquad \frac{A \vdash \&_{\ell \in L}\{\ell : B_\ell\} \quad (k \in L)}{A \vdash B_k}\ \&\textsc{e}$$

Theorem 8.8. *If $A \vdash B$ in natural deduction, then $A \vdash B$ in the sequent calculus.*

Proof.

$$\frac{A \vdash \&_{\ell \in L}\{\ell : B_\ell\} \quad (k \in L)}{A \vdash B_k}\ \&\textsc{e} \rightsquigarrow \frac{A \vdash \&_{\ell \in L}\{\ell : B_\ell\} \quad \dfrac{(k \in L)}{\&_{\ell \in L}\{\ell : B_\ell\} \vdash B_k}\ \&\textsc{l}}{A \vdash B_k}\ \textsc{cut}$$

□

Theorem 8.9. *If $A \vdash B$ in sequent calculus, then $A \vdash B$ in the natural deduction.*

Proof.

$$\frac{A_k \vdash C \quad (k \in L)}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash C}\ \&\textsc{l}$$

$$\rightsquigarrow$$

$$\frac{\dfrac{}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash \&_{\ell \in L}\{\ell : A_\ell\}}\ \textsc{hyp} \quad (k \in L)}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash A_k}\ \&\textsc{e} \qquad A_k \vdash C}{\&_{\ell \in L}\{\ell : A_\ell\} \vdash C}\ \textsc{subst}$$

$$\frac{\forall \ell \in L : A_\ell \vdash C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash C}\ \oplus\textsc{l}$$

$$\rightsquigarrow$$

$$\frac{\dfrac{}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \textsc{hyp} \qquad \forall \ell \in L : A_\ell \vdash C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash C}\ \oplus\textsc{e}$$

□

$$\frac{}{A{\downarrow} \vdash A{\downarrow}}\ \textsc{hyp} \qquad \frac{A{\downarrow} \vdash B{\downarrow} \quad B{\downarrow} \vdash C{\uparrow}}{A{\downarrow} \vdash C{\uparrow}}\ \textsc{subst} \qquad \frac{A{\downarrow} \vdash B{\downarrow} \quad B{\downarrow} \vdash C{\downarrow}}{A{\downarrow} \vdash C{\downarrow}}\ \textsc{subst}$$

$$\frac{A{\downarrow} \vdash B_k{\uparrow} \quad (k \in L)}{A{\downarrow} \vdash \oplus_{\ell \in L}\{\ell : B_\ell\}{\uparrow}}\ \oplus\textsc{i} \qquad \frac{A{\downarrow} \vdash \oplus_{\ell \in L}\{\ell : B_\ell\}{\uparrow} \quad \forall \ell \in L : B_\ell{\downarrow} \vdash C{\uparrow}}{A{\downarrow} \vdash C{\uparrow}}\ \oplus\textsc{e}$$

$$\frac{\forall \ell \in L : A{\downarrow} \vdash B_\ell{\uparrow}}{A{\downarrow} \vdash \&_{\ell \in L}\{\ell : B_\ell\}{\uparrow}}\ \&\textsc{i} \qquad \frac{A{\downarrow} \vdash \&_{\ell \in L}\{\ell : B_\ell\}{\downarrow} \quad (k \in L)}{A{\downarrow} \vdash B_k{\downarrow}}\ \&\textsc{e}$$

8.9 *Connections to Basic Logic*

Sambin et al. (2000) proposes Basic Logic in which connectives are defined by a single *definitional equation*. 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.

$$\frac{\Omega \vdash A \quad \Omega \vdash B}{\Omega \vdash A \mathbin{\&} B}\ \&\textsc{f} \qquad \frac{\Omega \vdash A \mathbin{\&} B}{\Omega \vdash A}\ \&\textsc{ir}_1 \qquad \frac{\Omega \vdash A \mathbin{\&} B}{\Omega \vdash B}\ \&\textsc{ir}_2$$

By trivializing the implicit reflection rules, Sambin et al. arrive at the axioms

$$\frac{}{A \mathbin{\&} B \vdash A}\ \&\textsc{a}_1 \qquad \frac{}{A \mathbin{\&} B \vdash B}\ \&\textsc{a}_2$$

Combining these axioms with CUT, they arrive at *explicit reflection* rules:

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

$$\frac{\dfrac{}{A \mathbin{\&} B \vdash A}\ \&\textsc{a}_1 \qquad \Omega'_L\, A\, \Omega'_R \vdash C}{\Omega'_L\, (A \mathbin{\&} B)\, \Omega'_R \vdash C}\ \textsc{cut}^A$$

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.

$$\frac{\Omega \vdash A \mathbin{\&} B \qquad \dfrac{}{A \mathbin{\&} B \vdash A}\ \&\textsc{a}_1}{\Omega \vdash A}\ \textsc{cut}^{A \mathbin{\&} B}$$

As ANOTHER EXAMPLE of this process, consider the definitional equation for ordered conjunction:

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

The formation and implicit reflection rules are:

$$\frac{\Omega'_L\, A\, B\, \Omega'_R \vdash C}{\Omega'_L\, (A \bullet B)\, \Omega'_R \vdash C}\ \bullet\textsc{f} \qquad \frac{\Omega'_L\, (A \bullet B)\, \Omega'_R \vdash C}{\Omega'_L\, A\, B\, \Omega'_R \vdash C}\ \bullet\textsc{ir}$$

By trivializing the implicit reflection rule, we obtain the axiom

$$\frac{}{A\,B \vdash A \bullet B}\ \bullet\text{A}$$

Combining this axiom with CUT, we arrive at the explicit reflection rule

$$\frac{\Omega_1 \vdash A \quad \Omega_2 \vdash B}{\Omega_1\,\Omega_2 \vdash A \bullet B}\ \bullet\text{ER}$$

$$\dfrac{\Omega_1 \vdash A \quad \dfrac{\Omega_2 \vdash B \quad \dfrac{}{A\,B \vdash A \bullet B}\ \bullet\text{A}}{A\,\Omega_2 \vdash A \bullet B}\ \text{CUT}^{B}}{\Omega_1\,\Omega_2 \vdash A \bullet B}\ \text{CUT}^{A}$$

In the other direction, the axiom can also be obtained by trivializing the explicit reflection rule; the implicit reflection rules is then obtained from CUT.

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

As a final example of this process, consider the definitional equation for left-handed implication:

$$\frac{A\,\Omega \vdash B}{\Omega \vdash A \setminus B}\ \setminus$$

The formation and implicit reflection rules are:

$$\frac{A\,\Omega \vdash B}{\Omega \vdash A \setminus B}\ \setminus\text{F} \qquad \frac{\Omega \vdash A \setminus B}{A\,\Omega \vdash B}\ \setminus\text{IR}$$

By trivializing the implicit reflection rule, we obtain the axiom

$$\frac{}{A\,(A \setminus B) \vdash B}\ \setminus\text{A}$$

Combining this axiom with CUT, we arrive at the explicit reflection rule

$$\frac{\Omega \vdash A \quad \Omega'_L\,B\,\Omega'_R \vdash C}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \setminus\text{ER}$$

$$\dfrac{\Omega \vdash A \quad \dfrac{\dfrac{}{A\,(A \setminus B) \vdash B}\ \setminus\text{A} \quad \Omega'_L\,B\,\Omega'_R \vdash C}{\Omega'_L\,A\,(A \setminus B)\,\Omega'_R \vdash C}\ \text{CUT}^{B}}{\Omega'_L\,\Omega\,(A \setminus B)\,\Omega'_R \vdash C}\ \text{CUT}^{A}$$

In the other direction, the axiom can also be obtained by trivializing the explicit reflection rule; the implicit reflection rules is then obtained from CUT.

$$\dfrac{\Omega \vdash A \setminus B \quad \dfrac{}{A\,(A \setminus B) \vdash B}\ \setminus\text{A}}{A\,\Omega \vdash B}\ \text{CUT}^{A \setminus B}$$

9
A computational interpretation of the singleton Hilbert system as session-typed communicating chains

In the previous chapter, we took a purely proof-theoretic view of singleton logic and its sequent calculus and Hilbert-style axiomatization. The proof terms assigned to Hilbert-style proofs were simply syntactic objects, and the proof of non-analytic cut elimination(??) 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 too strong to ignore: We saw that the principal cases in the [proof of] admissibility of non-analytic cuts(??) 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 Hilbert-style axiomatization of singleton logic.

$$\underset{\rightarrow}{k} \bullet \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$$
$$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \bullet \underset{\leftarrow}{k} = N_k$$

In particular, we will see that singleton propositions can be interpreted as session types that describe patterns of interprocess communication(??); Hilbert-style proofs, as chains of session-typed processes(??); and cut reduction, as asynchronous message-passing communication(??).[1] 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.

[1] As we will see in ??, Hilbert-style proofs may also be viewed as well-behaved chains of the communicating automata familiar from ??.

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[2] – with two key differences. First, unlike SILL, we use singleton logic, not intuitionistic linear logic. Second and most importantly, we use a Hilbert-style axiomatization, rather than a standard sequent calculus like SILL does. The use of Hilbert-style 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]

[2] Caires et al. 2016, 2012.

[3] It is possible to give a rather ad hoc treatment of asynchronous communication using SILL's sequent proofsDeYoung et al. (2012), but, in our view, the treatment of asynchronous communication using Hilbert-style proofs is far more elegant.

We begin, in ??, by introducing process chains as ...

9.1 *Process chains and process expressions*

9.1.1 *Untyped process chains*

[By analogy with chains of communicating automata,] 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 linear 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 linear topology, the channels need not be named – they can instead be referred to as simply the left- and right-hand channels of P_i.

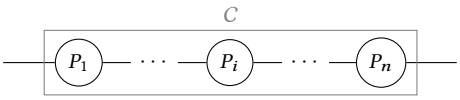


Figure 9.1: A prototypical process chain, C

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 [surroundings], 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 the monoid operation, end-to-end composition of chains, and $\cdot$ denotes the empty chain. As a monoid, chains are subject to associativity and unit laws:

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$(\cdot) \parallel C = C = C \parallel (\cdot)$$

We do not distinguish chains that are equivalent up to these laws, instead treating such chains as syntactically identical.

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.

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 ... by a process expression P. Processes are thus dynamic realizations of the static

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

process expressions, in the same way that executables run source code. It is sometimes convenient to blur

Session types describe the patterns by which a process is permitted to communicate with its left- and right-hand neighbors.

To follow a Curry–Howard isomorphism, we will adopt the propositions as session types; Hilbert-style proof terms as process expressions; and Hilbert system's inference rules as session-typing rules.

We will reinterpret the proof-term judgment of the singleton Hilbert system as 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 concurrent 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.

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, additive disjunction is interpreted as *internal choice*. The service ... is The internal choice $\oplus_{\ell \in L}\{\ell : A_\ell\}$ is the service in which the provider chooses which one of the services $(A_\ell)_{\ell \in L}$ it will offer its client.

type of a process that sends some label $k \in L$ to its right-hand client and then behaves like A_k.

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 interpreted 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 $\oplus\mathrm{R}'$ and $\oplus\mathrm{L}$ rules:

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\mathrm{R}' \qquad \frac{\forall \ell \in L\colon\ \ A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\mathrm{L}$$

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. [Dually,] 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.

Dually, additive conjunction becomes a form of external choice. A provider of service $\&_{\ell \in L}\{\ell : A_\ell\}$ offers its client its choice of services $(A_\ell)_{\ell \in L}$.

$$\frac{\forall \ell \in L\colon\ \ A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\mathrm{R} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\mathrm{L}'$$

The client is a message $\underleftarrow{k}$ that uses its payload [of label $k \in L$] to indicate the client's choice. The provider, $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$, is an input process that awaits a message indicating the client's choice and then continues along the chosen branch.

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.

$$\frac{\forall \ell \in L\colon\ \ A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\mathrm{R} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\mathrm{L}'$$

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.

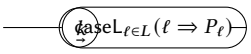
The proof term $P_1 \diamond P_2$ for composition of proofs is now reinterpreted as the expression for a process that will spawn new, neighboring threads of control for P_1 and P_2 and then terminate the original thread of control. In effect, $P_1 \diamond P_2$ now composes process [behaviors].

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \mathrm{CUT}^B$$

For $P_1 \diamond P_2$ to be a well-typed composition, the communication protocol

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.

$$\frac{}{A \vdash \leftrightarrow : A}\ \mathrm{ID}^A$$



Process expressions, P, and their session-typing rules are isomorphic to the Hilbert-style proof terms and inference rules of ??. [Propositions are reinterpreted as session types.]

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 individual processes, a chain C thus enjoys client and provider relationships with its left- and right-hand environments, respectively.

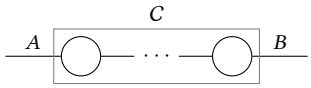


Figure 9.2: A well-typed process chain that uses service A to offer service B

The simplest session-typing rule for chains is the one that types a chain consisting of a single running process P:

$$\frac{A \vdash P : B}{A \Vdash P : B}\ \textsc{c-proc}$$

In other words, a running process has the same type as its underlying process expression.

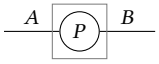


Figure 9.3: A chain made of one well-typed process that uses service A to offer service B

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:

$$\frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A$$

Save for the contrasting $\Vdash$ turnstile and the empty chain in place of a forwarding process expression, this mirrors the identity principle ... We will see that this is not a coincidence.

Figure 9.4: A well-typed empty chain that uses service A to offer service A

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:

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B$$

Once again, there are strong similarities to a process expression, in this case $P_1 \diamond P_2$ and its $\textsc{cut}^B$ session-typing rule. We can make these similarities explicit by defining a homomorphism, $(-)^\sharp$, from chains to process expressions: This function is type-preserving:

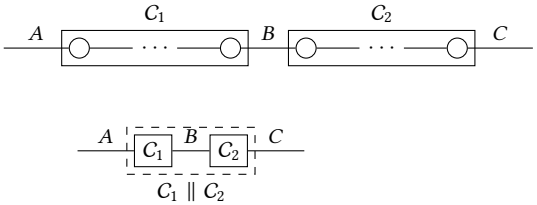


Figure 9.5: A well-typed process chain that uses service A to offer service B

$$(C_1 \parallel C_2)^\sharp = C_1^\sharp \diamond C_2^\sharp$$
$$(\cdot)^\sharp = \leftrightarrow$$
$$P^\sharp = P$$

Figure 9.6: A homomorphism from chains to process expressions

THEOREM 9.1. *If $A \Vdash C : B$, then $A \vdash C^\sharp : B$.*

Proof. By structural induction on the session-typing derivation. □

Figure 9.7: Well-typed process expressions and process chains

$$\text{SESSION TYPES} \quad A, B, C ::= \alpha \mid \oplus_{\ell \in L}\{\ell : A_\ell\} \mid \&_{\ell \in L}\{\ell : A_\ell\}$$

$$\text{PROCESS CHAINS} \quad \mathcal{C}, \mathcal{D} ::= (C_1 \parallel C_2) \mid \cdot \mid P$$

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$(\cdot) \parallel C = C = C \parallel (\cdot)$$

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B \qquad \frac{}{A \vdash \cdot : A}\ \textsc{c-id}^A \qquad \frac{A \vdash P : B}{A \Vdash P : B}\ \textsc{c-proc}$$

$$\text{PROCESS EXPRESSIONS} \quad P, Q ::= P_1 \diamond P_2 \mid \leftrightarrow \mid \underrightarrow{k} \mid \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \underleftarrow{k}$$

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \leftrightarrow : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L: \quad A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L: \quad A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\textsc{l}'$$

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 syntactically symmetric, but because we view them as (binary) hypothetical judgments, a judgmental asymmetry between the antecedent and consequent remains. It is this judgmental asymmetry that is reflected in the provider–client, offer–use asymmetry.

9.1.4 *From admissibility of non-analytic cuts to an operational semantics*

In the previous chapter, we presented a procedure for normalizing Hilbert-style [singleton?] proofs. Proof normalization was important to establish

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 ... The situation is analogous to that of intuitionistic natural deduction and simply-typed functional computation:

In fact, the difference is even starker here because, once recursive process definitions are introduced(??), 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 Hilbert-style proof normalization will still directly inform the operational semantics of processes.

- Operational semantics does not observe processes, observes only messages

In the previous ??, 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:

$$\frac{(k \in L)}{\underset{\rightarrow}{k} \parallel \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \longrightarrow P_k} \qquad \frac{(k \in L)}{\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel \underset{\leftarrow}{k} \longrightarrow P_k}$$

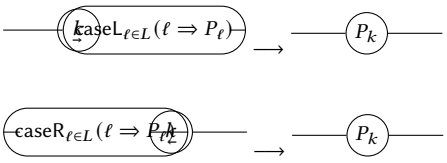


Figure 9.8: Pictorial representations of the principal reductions

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 normalization steps in the proof of admissibility of non-analytic cuts: $\underset{\rightarrow}{k} \blacklozenge \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$ and $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge \underset{\leftarrow}{k} = N_k$.

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 $\leftrightarrow$:

$$\frac{}{P_1 \diamond P_2 \longrightarrow P_1 \parallel P_2} \qquad \frac{}{\leftrightarrow \longrightarrow \cdot}$$

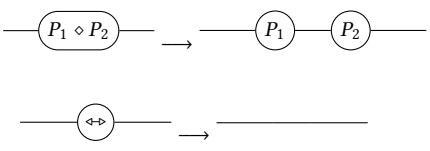


Figure 9.9: Pictorial representations of the reductions for $P_1 \diamond P_2$ and $\leftrightarrow$

Because process chains are always considered up to associativity and unit laws, these reduction rules (along with the above $\|$-compatibility rules) reflect the associative and identity normalization steps in the proof of admissibility of non-analytic cuts(??). For example, just as

$$(N_0 \diamond \underset{\rightarrow}{k}) \blacklozenge M = N_0 \blacklozenge (\underset{\rightarrow}{k} \blacklozenge M)$$

is an associative normalization step,

$$(P_0 \diamond \underset{\rightarrow}{k}) \parallel P_1 \longrightarrow\!= P_0 \parallel (\underset{\rightarrow}{k} \parallel P_1)$$

is a reduction. Similarly, $\leftrightarrow \parallel P \longrightarrow\!= P$ is a reduction that reflects the normalization step $\leftrightarrow \blacklozenge M = M$.

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 $\mathsf{caseL}()$:

$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \blacklozenge M = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell \blacklozenge M)\,.$$

As part of proof normalization, this step is quite natural because it progresses toward a normal form by pushing the admissible cut ($\blacklozenge$ 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 brings the chain C, a dynamic object, within the static context of a process expression, $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow -)$.

THEOREM 9.2 (Preservation). *If $A \Vdash C : B$ and $C \longrightarrow C'$, then $A \Vdash C' : B$.*

THEOREM 9.3 (Progress). *If $A \Vdash C : B$, then:*

- *chain C can reduce, that is, $C \longrightarrow C'$;*
- *chain C is waiting to interact with its right-hand client, that is, $C = C' \parallel \underset{\rightarrow}{k}$ 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, $C = \underset{\leftarrow}{k} \parallel C'$ or $C = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel C'$; or*
- *chain C is empty, that is, $C = \cdot$.*

9.2 *Recursive type and process definitions*

- coinductively defined types; productivity = contractivity

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.

coinductive behavior. For instance, a transducer process that transforms a stream of input symbols into a stream of output symbols cannot be represented.

The solution is to introduce mutually recursive type definitions, in a manner reminiscent of the recursively defined ordered propositions, $\alpha \triangleq A$, seen in **??**. However, recursively defined types are not particularly useful if process expressions remain finitary, so we also introduce mutually recursive processes: $A \vdash p : C \triangleq P$. In both cases, the recursion is defined using general fixed points, not least or greatest fixed points.[4]

We require that recursive type definitions be *contractive*[5] – that the body of each recursive type definition begin with a type constructor ($\oplus$ or $\&$) at the top level. This rules out problematic definitions like $\alpha \triangleq \alpha$. Moreover, it justifies an *equi*recursive treatment of types in which type definitions may be silently unfolded (or folded) at will. In other words, a type $\alpha \triangleq A$ is equal to its unfolding, $[A/\alpha]A$.

This stands in contrast with an *iso*recursive treatment of types in which recursive types are constructed by a fixed point operator $\mu\alpha.A$.

We require contractivity of process definitions as well, ruling out definitions like $A \vdash p : C \triangleq p$. Like type definitions, process expressions are treated equirecursively and may be freely and silently unfolded (or folded).

$$\frac{}{\vdash -NoValue-}\ \oplus\text{L}$$

PROCESS EXPRESSIONS $P, Q ::= \cdots \mid p$

At this point, we must also choose whether to treat recursive type and process definitions *equi*recursively or *iso*recursively.

Once recursive type and process definitions are added, there is, strictly speaking, no longer a Curry–Howard isomorphism between session-typed process chains and the Hilbert-style proofs of singleton logic. Importantly, however, the core system remains unchanged and still enjoys the isomorphism because the recursion 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 constuctor for simple function types remains unchanged and still isomorphic with intuitionistic implication.

These types are defined using general fixed points, not least or greatest fixed points – the types are recursive but not inductive nor coinductive.

Example. *We may now adapt the previous example into a collection of process definitions*

$$\sigma \vdash q_0 : \sigma \triangleq \mathsf{caseL}(a \Rightarrow q_0 \diamond \underset{\rightarrow}{a} \mid b \Rightarrow q_1 \diamond \underset{\rightarrow}{b})$$
$$\sigma \vdash q_1 : \sigma \triangleq \mathsf{caseL}(a \Rightarrow q_1 \diamond \underset{\rightarrow}{a} \mid b \Rightarrow q_1)$$

9.3 *Automata and transducers*

$$\Sigma^\omega \vdash q : \Gamma^* \triangleq \mathsf{caseL}_{a \in \Sigma}(a \Rightarrow q'_a \diamond \underset{\rightarrow}{w}_{q,a})$$

[4] Treatments of inductive and coinductive types in linear logic (Baelde 2012; Toninho et al. 2014) should be adaptable to a singleton logic setting, and work on a Curry–Howard extrapolation of Fortier and Santocanale's work on circular proofsFortier and Santocanale (2013) is underway (Derakhshan and Pfenning 2019).

[5] Gay and Hole 2005.

9.4 *Toward asynchronous SILL*

Most work on SILL uses a synchronous interpretation of cut reductions as communication.

$$\dfrac{\dfrac{\Delta, y{:}A \vdash P :: x{:}B}{\Delta \vdash x(y).P :: x{:}A \multimap B}\ \multimap\!\mathrm{R} \qquad \dfrac{\Delta_1' \vdash Q_1 :: y{:}A \quad \Delta_2', x{:}B \vdash Q_2 :: z{:}C}{\Delta_1', \Delta_2', x{:}A \multimap B \vdash (\nu y)\overline{x}\langle y\rangle.(Q_1 \mid Q_2) :: z{:}C}\ \multimap\!\mathrm{L}}{\Delta_1', \Delta_2', \Delta \vdash (\nu x)(x(y).P \mid (\nu y)\overline{x}\langle y\rangle.(Q_1 \mid Q_2)) :: z{:}C}\ \mathrm{CUT}$$

$$\longrightarrow$$

$$\dfrac{\dfrac{\Delta_1' \vdash Q_1 :: y{:}A \quad \Delta, y{:}A \vdash P :: x{:}B}{\Delta, \Delta_1' \vdash (\nu y)(Q_1 \mid P) :: x{:}B}\ \mathrm{CUT} \qquad \Delta_2', x{:}B \vdash Q_2 :: z{:}C}{\Delta_2', \Delta, \Delta_1' \vdash (\nu x)((\nu y)(Q_1 \mid P) \mid Q_2) :: z{:}C}\ \mathrm{CUT}$$

$$\dfrac{\Delta, y{:}A \vdash P :: x'{:}B}{\Delta \vdash x(y, x').P :: x{:}A \multimap B}\ \multimap\!\mathrm{R} \qquad \dfrac{}{x{:}A \multimap B, y{:}A \vdash \overline{x}\langle y, x'\rangle :: x'{:}B}\ \multimap\!\mathrm{L}'$$

$$\dfrac{}{\Gamma, u{:}A; \cdot \vdash \overline{x}\langle u\rangle :: x{:}!A}\ !\mathrm{R}' \qquad \dfrac{\Gamma, u{:}A; \Delta \vdash P :: z{:}C}{\Gamma; \Delta, x{:}!A \vdash x(u).P :: z{:}C}\ !\mathrm{L}$$

$$\dfrac{}{\Gamma, u{:}A; \cdot \vdash \overline{x}\langle u\rangle :: x{:}!A}\ !\mathrm{R}' \qquad \dfrac{\Gamma, u{:}A; \Delta \vdash P :: z{:}C}{\Gamma; \Delta, x{:}!A \vdash x(u).P :: z{:}C}\ !\mathrm{L}$$

9.5

9.5.1 *Process chains*

By analogy with chains of communicating automata, 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 linear 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 internal state. Because process chains always maintain a linear topology, channels need not be named – they can instead be referred to as simply the left- and right-hand channels of P_i.

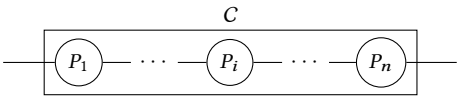


Figure 9.10: A prototypical process chain, C

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 [surroundings], they may be referred to as the left- and right-hand channels of the chain.

Chains may be composed end to end by conjoining the right-hand channel of one chain with the left-hand channel of another chain.

CHAINS AS A FREE MONOID Moving from this informal intuition to a more formal characterization, process chains C form a free monoid over processes P:

$$C ::= \cdot \mid (C_1 \parallel C_2) \mid P,$$

where $\cdot$ denotes the empty chain and $\parallel$ denotes the monoid operation, chain composition. As the monoid operation, composition is subject to the usual associativity and unit laws[6]:

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$\cdot \parallel C = C = C \parallel \cdot$$

[6] Unlike composition in most process calculi, chain composition is not commutative.

Because these monoid laws may be freely applied, we switch between two alternative views of process chains whenever convenient: the view that a chain C is either empty ($C = \cdot$), a composition ($C = C_1 \parallel C_2$), or a single process ($C = P$); and the view that a chain C is a finite sequence of processes ($C = P_1 \parallel \cdots \parallel P_n$).

To ..., a session-type system for process chains can be developed. to describe how the process chain C interacts with its environment, we use a judgment

$$A \Vdash C : B,$$

meaning that the chain C offers service B along its right-hand channel, while concurrently using service A along its left-hand channel.

For a chain composition $C_1 \parallel C_2$ to be well-typed, the service offered by C_1 along its right-hand channel must be the same service that C_2 expects to use along its left-hand channel. Otherwise, communication between C_1 and C_2

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B$$

The empty chain, $\cdot$, offers a service A to its right by directly using the same service from its left:

$$\frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A$$

Lastly, a chain that consists of a single process P is well-typed if its process expression P is well-typed:

$$\frac{A \vdash P : C}{A \Vdash P : C}\ \textsc{c-proc}$$

Offers/uses distinction: retained for consistency with the hypothetical judgement asymmetry and SILL. Judgmental asymmetry between antecedents and consequents of a sequent.

A chain C does not compute in isolation, but instead interacts with its environment along two channels: to its left along the left-hand channel of P_1, and to its right along the right-hand channel of P_n. Chains can be composed end to end by The left-hand channel of P_1 and the right-hand channel of

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B \qquad \frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A \qquad \frac{A \vdash P : C}{A \Vdash P : C}\ \textsc{c-proc}$$

Figure 9.11: Process chains and their session-type system

More formally, as ordered lists of processes, process chains form a free monoid.

Alternatively, process chains may be characterized algebraically as forming a free monoid over processes.

Process chains form a free monoid...

Process chains communicate with their environment...

The judgment $A \Vdash C : B$ describes the pattern of communication...

Chain composition

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B$$

Empty chain

$$\frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A$$

Monoid laws applies silently ...

Chain consisting of one process

$$\frac{A \vdash P : B}{A \Vdash P : B}\ \textsc{c-proc}$$

9.6 *Session-typed asynchronous process chains*

- Foreshadow theorem about relationship with communicating automata

By analogy with chains of communicating automata, we envision a process chain, C, as a finite sequence of processes $(P_i)_{i=1}^n$, each with its own independent thread of control and arranged in a linear topology. As depicted in the adjacent figure, each process P_i shares a unique channel with its left-hand neighbor and a unique channel with its right-hand neighbor. These channels need not be named – they can instead be referred to as simply the left- and right-hand channels of P_i.

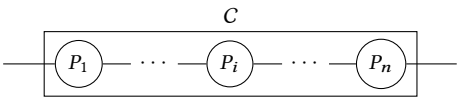


Figure 9.12: A process chain, C

Process chains are never isolated from the surrounding environment. Both the left-hand channel of P_1 and the right-hand channel of P_n continue to allow external communication, even as communication among neighboring processes changes the chain's internal state.

As a string of processes,

Formally, then, process chains C form a free monoid over processes P:

$$C ::= \cdot \mid (C_1 \parallel C_2) \mid P\,,$$

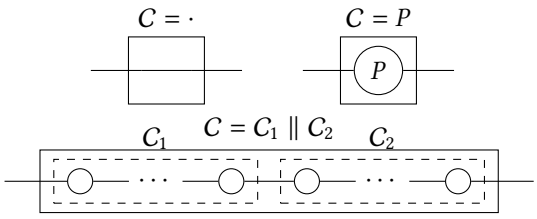


Figure 9.13: A graphical depiction of process chain constructors

SESSION TYPES $A ::= \alpha \mid \oplus_{\ell \in L}\{\ell : A_\ell\} \mid \&_{\ell \in L}\{\ell : A_\ell\}$

PROCESS EXPRESSIONS $P ::= P_1 \diamond P_2 \mid \leftrightarrow \mid \underrightarrow{k} \mid \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \mid \underleftarrow{k}$

$$\frac{A \vdash P_1 : B \quad B \vdash P_2 : C}{A \vdash P_1 \diamond P_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \leftrightarrow : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underrightarrow{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon\ A_\ell \vdash P_\ell : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon\ A \vdash P_\ell : C_\ell}{A \vdash \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) : \&_{\ell \in L}\{\ell : C_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : C_\ell\} \vdash \underleftarrow{k} : C_k}\ \&\textsc{l}'$$

Figure 9.14: Asynchronous process chains and their session-type system

where we write $\cdot$ for the empty chain and $\|$ for the monoid operation, which is subject to the usual associativity and unit laws:

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$\cdot \parallel C = C = C \parallel \cdot$$

Figure 9.13 gives a graphical depiction of the three basic shapes that chains may take.

So far, this definition of process chains has intentionally abstracted from what exactly a process is, and how exactly communication occurs over channels.

Figure 9.14 presents the syntax of process chains and their session-type system. Formally, the session types are identical to the propositions of singleton logic; the process terms, identical to the Hilbert-style proof terms; and the session-typing rules, identical to the Hilbert-style inference rules. In fact, the whole of this figure is identical to fig. 8.7, save for the small difference in terminology.

This size of this difference, however, belies its significance.

- The proof term $P_1 \diamond P_2$ for composition of proofs is now reinterpreted as the expression for a process that will spawn, to the immediate left, a new thread of control for P_1, while the original thread of control continues with P_2. In effect, $P_1 \diamond P_2$ now composes process behaviors.

Figure 9.15: Syntax and session-typing rules for process chains

$$\text{PROCESS CHAINS} \quad C ::= \cdot \mid (C_1 \parallel C_2) \mid P$$

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{cut}^B \qquad \frac{}{A \Vdash \cdot : A}\ \textsc{id}^A \qquad \frac{A \vdash P : C}{A \Vdash P : C}\ \textsc{proc}$$

$$(C_1 \parallel C_2) \parallel C_3 = C_1 \parallel (C_2 \parallel C_3)$$
$$\cdot \parallel C = C = C \parallel \cdot$$

- The proof term $\leftrightarrow$ is reinterpreted as the expression for a process that terminates its thread of control, excising the process from the chain.
- The proof terms $\underset{\leftarrow}{k}$ and $\underset{\rightarrow}{k}$ are now viewed as messages carrying the label k as their payloads. The direction of the underlying arrow indicates the message's intended recipient: $\underset{\leftarrow}{k}$ is being sent to the left-hand neighbor; $\underset{\rightarrow}{k}$, to the right-hand neighbor.
- The proof term $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is reinterpreted as the expression for a process that waits to receive a message $\underset{\rightarrow}{k}$ from its left-hand neighbor and then branches on the received label, so that the thread of control continues with P_k. The proof term $\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$ is interpreted dually as the expression for a process that branches on a message from its right-hand neighbor.

Just as session types characterize the communication behavior of individual process expressions, the same types can be used to describe the behavior of entire process chains. The judgment $A \Vdash C : B$ indicates that the process chain C is well-typed, with the left-hand channel of C having type A and the right-hand channel having type B.

The simplest chain is the one that consists of a single process P; the chain inherits the process's type:

$$\frac{A \vdash P : C}{A \Vdash P : C}\ \textsc{c-proc}$$

The composition $C_1 \parallel C_2$ is typable if the two chains assign the same type to their shared channel.

$$\frac{A \Vdash C_1 : B \quad B \Vdash C_2 : C}{A \Vdash C_1 \parallel C_2 : C}\ \textsc{c-cut}^B$$

$$\frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A$$

Chains can be reified as process expressions. Let $(-)^\sharp$ be a function from chains to process expressions given by

$$(\cdot)^\sharp = \leftrightarrow$$
$$(C_1 \parallel C_2)^\sharp = C_1^\sharp \diamond C_2^\sharp$$
$$P^\sharp = P$$

THEOREM 9.4. *If $A \Vdash C : B$, then $A \vdash C^\sharp : B$.*

9.6.1 *From admissibility of non-analytic cuts to an operational semantics*

In the previous chapter, we presented a procedure for normalizing Hilbert-style [singleton?] proofs. Full proof normalization was important to ...

In this chapter, however, our perspective has shifted from proof theory to concurrent computation, from proofs to processes. And so full normalization is no longer appropriate – we now want to expose the concurrent computational behavior, not just ... The situation is analogous to that of intuitionistic natural deduction and simply-typed functional computation:

In fact, the difference is even starker here because, once recursive process definitions are introduced(??), 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 Hilbert-style proof normalization will still directly inform the operational semantics of processes.

- Operational semantics does not observe processes, observes only messages

IN THE PREVIOUS ??, 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.

At the heart of the operational semantics for process chains is *reduction*, a binary relation on chains which we write as $\longrightarrow$. Reductions may occur among any of the chain's processes, and thus the relation is compatible with the monoid operation, $\parallel$:

$$\frac{C_1 \longrightarrow C_1'}{C_1 \parallel C_2 \longrightarrow C_1' \parallel C_2} \qquad \frac{C_2 \longrightarrow C_2'}{C_1 \parallel C_2 \longrightarrow C_1 \parallel C_2'}$$

A process $P_1 \diamond P_2$ spawns, to its immediate left, a new thread of control for P_1, while the original thread of control continues with P_2.

$$\frac{}{P_1 \diamond P_2 \longrightarrow P_1 \parallel P_2}$$

Because process chains are always ... up to associativity and unit laws, these reductions

Recall

$$(N_0 \diamond \underrightarrow{k}) \bullet M = N_0 \bullet (\underrightarrow{k} \bullet M)$$
$$N \bullet (\underleftarrow{k} \diamond M_0) = (N \bullet \underleftarrow{k}) \bullet M_0$$

$$\leftrightarrows \bullet M = M$$
$$N \bullet \leftrightarrows = N$$

$$\underrightarrow{k} \bullet \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow M_\ell) = M_k$$
$$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N_\ell) \bullet \underleftarrow{k} = N_k$$

$$(\underleftarrow{k} \diamond N_0) \bullet M = \underleftarrow{k} \diamond (N_0 \bullet M)$$
$$N \bullet (M_0 \diamond \underrightarrow{k}) = (N \bullet M_0) \diamond \underrightarrow{k}$$
$$\underleftarrow{k} \bullet M = \underleftarrow{k} \diamond M$$
$$N \bullet \underrightarrow{k} = N \diamond \underrightarrow{k}$$
$$\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell) \bullet M = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow N_\ell \bullet M)$$
$$N \bullet \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow M_\ell) = \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow N \bullet M_\ell)$$

etc.

- The operational semantics uses a particular strategy: $\longrightarrow$ is the least compatible relation that satisfies the following.

$$P_1 \diamond P_2 \longrightarrow P_1 \parallel P_2$$
$$\leftrightarrows \longrightarrow \cdot$$
$$\underrightarrow{k} \parallel \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \longrightarrow P_k$$
$$\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel \underleftarrow{k} \longrightarrow P_k$$

We denote the reflexive, transitive closure of $\longrightarrow$ by $\Longrightarrow$.

THEOREM 9.5 (Type preservation). *If $A \Vdash C : B$ and $C \longrightarrow C'$, then $A \Vdash C' : B$.*

Proof. By structural induction on the given chain. □

LEMMA 9.6. *If $A \Vdash C : B$ and $C = C'$, then $A \Vdash C' : B$.*

THEOREM 9.7 (Progress). *If $A \Vdash C : B$, then either:*

- *$C \longrightarrow C'$ for some C';*
- *C is empty: $C = \cdot$;*
- *C is ready to communicate along its left-hand channel: $C = \underleftarrow{k} \parallel C_0$ or $C = \mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \parallel C_0$ for some C_0; or*
- *C is ready to communicate along its right-hand channel: $C = C_0 \parallel \underrightarrow{k}$ or $C = C_0 \parallel \mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)$ for some C_0.*

Proof. By structural induction on the given process chain. □

THEOREM 9.8. *$C^\sharp \Longrightarrow C$ for all C.*

Proof. By structural induction on the given chain. □

EXAMPLE 9.1. An expression for a process that will wait for an a- or b-message to arrive from its left-hand neighbor and then send to its right-hand neighbor either two consecutive a-messages or a single b-message, respectively, is:

$$\oplus\{a : \epsilon, b : \epsilon\} \vdash \mathsf{caseL}(a \Rightarrow \underrightarrow{a} \diamond \underrightarrow{a} \mid b \Rightarrow \underrightarrow{b}) : \oplus\{a : \oplus\{a : \epsilon\}, b : \epsilon\}\,.$$

Indeed, the process chain in which that process is sent an a-message computes as follows.

$$\underrightarrow{a} \parallel \mathsf{caseL}(a \Rightarrow \underrightarrow{a} \diamond \underrightarrow{a} \mid b \Rightarrow \underrightarrow{b}) \longrightarrow \underrightarrow{a} \diamond \underrightarrow{a} \longrightarrow \underrightarrow{a} \parallel \underrightarrow{a}$$

□

Part IV

Comparing the two approaches

10

From processes to rewriting

10.1 *A shallow embedding of processes in ordered rewriting*

[1]

[1] Is this a shallow embedding? Is HOAS deep or shallow?

$$
\begin{aligned}
[\![P \diamond Q]\!]^+ &= [\![P]\!]^+ \bullet [\![Q]\!]^+ \\
[\![\leftrightarrow]\!]^+ &= \mathbf{1} \\
[\![\underset{\rightarrow}{k}]\!]^+ &= \underset{\rightarrow}{k} \\
[\![\underset{\leftarrow}{k}]\!]^+ &= \underset{\leftarrow}{k} \\
[\![P]\!]^+ &= \downarrow [\![P]\!]^- \\
[\![\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow Q_\ell)]\!]^- &= \mathop{\&}_{\ell \in L} (\underset{\leftarrow}{\ell} \setminus \uparrow [\![Q_\ell]\!]^+) \\
[\![\mathsf{caseR}_{\ell \in L}(\ell \Rightarrow P_\ell)]\!]^- &= \mathop{\&}_{\ell \in L} (\uparrow [\![P_\ell]\!]^+ / \underset{\rightarrow}{\ell}) \\
[\![P]\!]^- &= \uparrow [\![P]\!]^+
\end{aligned}
$$

$$
\begin{aligned}
[\![\cdot]\!] &= \cdot \\
[\![C_1 \parallel C_2]\!] &= [\![C_1]\!]\,[\![C_2]\!] \\
[\![P]\!] &= [\![P]\!]^+
\end{aligned}
$$

Theorem 10.1. *$[\![\,]\!]$ constitutes a (strong) reduction bisimulation. That is:*

- *If $C \longrightarrow C'$, then $[\![C]\!] \longrightarrow [\![C']\!]$.*
- *If $[\![C]\!] = \Omega \longrightarrow \Omega'$, then $C \longrightarrow C'$ for some C' such that $[\![C']\!] = \Omega'$.*

Notice that neither this ?? nor a corresponding weak reduction bisimulation ?? would hold if the unfocused form of ordered rewriting were used.[2]

[2] For example, $[\![\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell)]\!] = \mathop{\&}_{\ell \in L}(\underset{\leftarrow}{\ell} \setminus [\![Q_\ell]\!]) \longrightarrow (\underset{\rightarrow}{k} \setminus [\![Q_k]\!])$ if $k \in L$, but there is no configuration C' such that $\mathsf{caseL}_{\ell \in L}(\ell \Rightarrow P_\ell) \Longrightarrow C'$ and $[\![C']\!] = \underset{\rightarrow}{k} \setminus [\![Q_k]\!]$.

10.2 *A session type system for ordered rewriting*

$$\frac{A \vdash A_1^+ : B \quad B \vdash A_2^+ : C}{A \vdash A_1^+ \bullet A_2^+ : C}\ \textsc{cut}^B \qquad \frac{}{A \vdash \mathbf{1} : A}\ \textsc{id}^A$$

$$\frac{(k \in L)}{A_k \vdash \underset{\rightarrow}{k} : \oplus_{\ell \in L}\{\ell : A_\ell\}}\ \oplus\textsc{r}' \qquad \frac{\forall \ell \in L\colon\ A_\ell \vdash A_\ell^+ : C}{\oplus_{\ell \in L}\{\ell : A_\ell\} \vdash \downarrow \&_{\ell \in L}(\underset{\rightarrow}{\ell} \setminus \uparrow A_\ell^+) : C}\ \oplus\textsc{l}$$

$$\frac{\forall \ell \in L\colon\ A \vdash A_\ell^+ : B_\ell}{A \vdash \downarrow \&_{\ell \in L}(A_\ell^+ \,/\, \underset{\leftarrow}{\ell}) : \&_{\ell \in L}\{\ell : B_\ell\}}\ \&\textsc{r} \qquad \frac{(k \in L)}{\&_{\ell \in L}\{\ell : B_\ell\} \vdash \underset{\leftarrow}{k} : B_k}\ \&\textsc{l},$$

$$\frac{A \Vdash \Omega_1 : B \quad B \Vdash \Omega_2 : C}{A \Vdash \Omega_1\ \Omega_2 : C}\ \textsc{c-cut}^B \qquad \frac{}{A \Vdash \cdot : A}\ \textsc{c-id}^A \qquad \frac{A \vdash A^+ : B}{A \Vdash A^+ : B}\ \textsc{c-proc}$$

[3]

[3] Careful with the 0-ary forms because you could end up with $\mathbf{0} \vdash \top : C$ and *also* $A \vdash \top : \top$.

Theorem 10.2. *If $A \vdash P : B$, then $A \vdash [\![P]\!]^+ : B$. Conversely, if $A \vdash A^+ : B$, then $A \vdash P : B$ for some process P such that $[\![P]\!]^+ = A^+$.*

Similarly, if $A \Vdash C : B$, then $A \Vdash [\![C]\!] : B$. Conversely, if $A \Vdash \Omega : B$, then $A \Vdash C : B$ for some configuration C such that $[\![C]\!] = \Omega$.

10.3 *Examples*

$$\hat{q} \triangleq \mathsf{caseL}_{a \in \Sigma}(a \Rightarrow \hat{q}'_a \mid \epsilon \Rightarrow \hat{F}(q))$$

where $\hat{F}(q) = \underset{\rightarrow}{?}$ if $q \in F$ and $\hat{F}(q) = \underset{\rightarrow}{?}$ if $q \notin F$.

$$\begin{aligned}
[\![\hat{q}]\!]^- &\triangleq [\![\mathsf{caseL}_{a \in \Sigma}(a \Rightarrow \hat{q}'_a \mid \epsilon \Rightarrow \hat{F}(q))]\!]^- \\
&= \mathop{\&}_{a \in \Sigma}(\underset{\rightarrow}{a} \setminus \uparrow [\![\hat{q}'_a]\!]^+) \,\&\, (\underset{\leftarrow}{\epsilon} \setminus \uparrow [\![\hat{F}(q)]\!]^+) \\
&= \mathop{\&}_{a \in \Sigma}(\underset{\rightarrow}{a} \setminus \uparrow\downarrow [\![\hat{q}'_a]\!]^-) \,\&\, (\underset{\leftarrow}{\epsilon} \setminus \uparrow [\![\hat{F}(q)]\!]^+)
\end{aligned}$$

$$e \triangleq (e \bullet b_1 \,/\, \underset{\rightarrow}{i}) \,\&\, (\underset{\leftarrow}{z} \,/\, \underset{\leftarrow}{d})$$

$$\begin{aligned}
[\![e]\!]^- &\triangleq (\uparrow(\downarrow [\![e]\!]^- \bullet \downarrow [\![b_1]\!]^-) \,/\, \underset{\rightarrow}{i}) \,\&\, (\uparrow \underset{\leftarrow}{z} \,/\, \underset{\leftarrow}{d}) && e \triangleq (e \bullet b_1 \,/\, \underset{\rightarrow}{i}) \,\&\, (\underset{\leftarrow}{z} \,/\, \underset{\leftarrow}{d}) \\
[\![b_0]\!]^- &\triangleq (\uparrow\downarrow [\![b_1]\!]^- \,/\, \underset{\rightarrow}{i}) \,\&\, (\uparrow(\underset{\leftarrow}{d} \bullet \downarrow [\![b'_0]\!]^-) \,/\, \underset{\leftarrow}{d}) && b_0 \triangleq (\uparrow\downarrow b_1 \,/\, \underset{\rightarrow}{i}) \,\&\, (\underset{\leftarrow}{d} \bullet b'_0 \,/\, \underset{\leftarrow}{d}) \\
[\![b_1]\!]^- &\triangleq (\uparrow(\underset{\rightarrow}{i} \bullet \downarrow [\![b_0]\!]^-) \,/\, \underset{\rightarrow}{i}) \,\&\, (\uparrow(\downarrow [\![b_0]\!]^- \bullet \underset{\rightarrow}{s}) \,/\, \underset{\leftarrow}{d}) && b_1 \triangleq (\underset{\rightarrow}{i} \bullet b_0 \,/\, \underset{\rightarrow}{i}) \,\&\, (b_0 \bullet \underset{\rightarrow}{s} \,/\, \underset{\leftarrow}{d}) \\
[\![b'_0]\!]^- &\triangleq (\underset{\leftarrow}{z} \setminus \uparrow \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus \uparrow(\downarrow [\![b_1]\!]^- \bullet \underset{\rightarrow}{s})) && b'_0 \triangleq (\underset{\leftarrow}{z} \setminus \underset{\rightarrow}{z}) \,\&\, (\underset{\rightarrow}{s} \setminus b_1 \bullet \underset{\rightarrow}{s})
\end{aligned}$$

10.4

$$\left\langle \frac{(q \xrightarrow{a} q'_a)}{a\,\hat{q} \longrightarrow \hat{q}'_a} \right\rangle_{(q,a) \in Q \times \Sigma} \quad \text{and} \quad \left\langle \overline{\epsilon\,\hat{q} \longrightarrow \hat{F}(q)} \right\rangle_{q \in Q}$$

Either $\underset{\rightarrow}{a}\,\hat{q}$ or $a\,\underset{\leftarrow}{\hat{q}}$.

$$a \triangleq \mathop{\&}_{q \in Q} (\underset{\leftarrow}{q'_a} \,/\, \underset{\leftarrow}{q})$$
$$\epsilon \triangleq \mathop{\&}_{q \in Q} (\hat{F}(q) \,/\, \underset{\leftarrow}{q})$$

Previously, we argued that the state-oriented encoding required adequacy to be judged up to bisimilarity: $\underset{\rightarrow}{a}\,\hat{q} \longrightarrow \hat{q}'$ did not imply $q \xrightarrow{a} q'$ because the equivirecursive treatment of definitions means that the encoding is not injective.

In the symbol-oriented encoding, adequacy no longer needs to be judged up to bisimilarity: $a\,\underset{\leftarrow}{q} \longrightarrow \underset{\leftarrow}{q'}$ does indeed imply $q \xrightarrow{a} q'$. By inversion on the given rewriting, it suffices to show that $q \xrightarrow{a} q'_a$ and $\underset{\leftarrow}{q'_a} = \underset{\leftarrow}{q'}$ together imply $q \xrightarrow{a} q'$. This time, because states are encoded as atoms, which have an uncomplicated notion of equality, the encoding is injective.

These differences in equality of atoms and recursively defined propositions should perhaps not be unexpected given the correspondence between atoms and messages; recursively defined propositions and processes. Being observable, messages are easy to compare for equality. But processes' internal structures are hidden, and therefore it shouldn't be possible to compare them for equality,

We could try to reverse engineer an equivalence on input symbols from the rewriting bisimilarity of their encodings. The encodings of symbols a and b are bisimilar if, and only if:

- $q \xrightarrow{a} q'_a$ implies $q \xrightarrow{b} q'_a$, for all q and q'_a; and
- $q \xrightarrow{b} q'_b$ implies $q \xrightarrow{a} q'_b$, for all q and q'_b.

In other words, symbols a and b have bisimilar encodings exactly when those encodings are equal.

$\underset{\rightarrow}{\epsilon}\,\underset{\rightarrow}{w^{\mathrm{R}}}\,\hat{q} \cong \epsilon\, w^{\mathrm{R}}\,\underset{\leftarrow}{q}$ for all $w \in \Sigma^*$ and $q \in Q$.

$$\frac{}{\underset{\rightarrow}{\epsilon}\,\hat{q}\ \mathcal{R}\ \hat{F}(q)} \qquad \frac{\Omega\,\hat{q}'_a\ \mathcal{R}\ A^+ \quad (q \xrightarrow{a} q'_a)}{\Omega\,\underset{\rightarrow}{a}\,\hat{q}\ \mathcal{R}\ A^+} \qquad \frac{\Omega\ \mathcal{R}\ \mathbf{1}}{\Omega\ \mathcal{R}\ \cdot}$$

$$\frac{}{\epsilon\,\underset{\leftarrow}{q}\ \mathcal{S}\ \hat{F}(q)} \qquad \frac{\Delta\,\underset{\leftarrow}{q'_a}\ \mathcal{S}\ A^+ \quad (q \xrightarrow{a} q'_a)}{\Delta\,a\,\underset{\leftarrow}{q}\ \mathcal{S}\ A^+} \qquad \frac{\Omega\ \mathcal{S}\ \mathbf{1}}{\Omega\ \mathcal{S}\ \cdot}$$

The relation $\mathcal{R}\mathcal{S}^{-1} \cup \mathcal{R} \cup \mathcal{S}^{-1}$ is a labled bisimulation up to reflexivity.

10.5

Bibliography

Aranda, Jesús, Cinzia Di Giusto, Catuscia Palamidessi, and Frank D. Valencia (2007). "On Recursion, Replication and Scope Mechanisms in Process Calculi". In: *Formal Methods for Components and Objects, 5th International Symposium* (Amsterdam, The Netherlands, Nov. 7–10, 2006). Ed. by Frank S. de Boer, Marcello M. Bonsangue, Susanne Graf, and Willem P. de Roever. Vol. 4709. Lecture Notes in Computer Science. Springer, pp. 185–206 (cit. on pp. 64, 89).

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. 169).

Baelde, David (2012). "Least and Greatest Fixed Points in Linear Logic". In: *ACM Trans. Comput. Logic* 13.1 (Jan. 2012), 2:1–2:44 (cit. on p. 189).

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. 17, 181).

— (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 p. 181).

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. 17, 18, 82).

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, Department of Computer Science, Dec. 2003 (cit. on p. 32).

Derakhshan, Farzaneh and Frank Pfenning (2019). "Circular Proofs as Session-Typed Processes. A Local Validity Condition". In: *Logical Methods in Computer Science* (Aug. 6, 2019). Submitted (cit. on p. 189).

DeYoung, Henry, Luís Caires, Frank Pfenning, and Bernardo Toninho (2012). "Cut Reduction in Linear Logic as Asynchronous Session-Typed Commu-

nication". 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 p. 181).

Dummett, Michael (1976). *The William James Lectures*. Later published as: **Dummett:WJ76-HUP91** (cit. on pp. 35, 165).

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 p. 189).

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. 64, 90, 189).

Gentzen, Gerhard (1935). "Untersuchungen über das logische Schließen". In: *Mathematische Zeitschrift* 39, pp. 176–210, 405–431. English translation in: **Szabo:NH69** (cit. on pp. 35, 165).

Girard, Jean-Yves (1987). "Linear Logic". In: *Theoretical Computer Science* 50.1, pp. 1–101 (cit. on pp. 17, 29, 30, 57).

Lambek, Joachim (1958). "The Mathematics of Sentence Structure". In: *The American Mathematical Monthly* 65.3, pp. 154–170 (cit. on pp. 19, 29, 57, 59, 81, 83).

Martin-Löf, Per (1983). *Siena Lectures*. Apr. 6–9, 1983. Transcript later published as: **Martin-Lof:Siena83-NJPL96** (cit. on pp. 35, 165).

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. 36).

— (2016). *Lecture Notes from 15-816: Substructural Logics*. Carnegie Mellon University, Computer Science Department, Aug. 30–Dec. 8, 2016. `w` (cit. on p. 29).

Polakow, Jeff and Frank Pfenning (1999). "Relating Natural Deduction and Sequent Calculus for Intuitionistic Non-Commutative Linear Logic". In: *Electronic Notes in Theoretical Computer Science* 20, pp. 449–466 (cit. on pp. 19, 29, 36, 37, 57).

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. 169, 179).

Steinberger, Florian (2011). "Why Conclusions Should Remain Single". In: *Journal of Philosophical Logic* 40.3 (June 2011), pp. 333–355 (cit. on p. 161).

Toninho, Bernardo, Luís Caires, and Frank Pfenning (2014). "Corecursion and Non-Divergence in Session-Typed Processes". In: *Trustworthy Global Com-*

puting. Ed. by Matteo Maffei and Emilio Tuosto. Springer, pp. 159–175 (cit. on p. 189).