Compositional Semantics for Shared-Variable Concurrency

ANONYMOUS AUTHOR(S)

This paper revisits the fundamental problem of defining a compositional semantics for a concurrent programming language under sequentially consistent memory with the aim of equating the denotations of pieces of code if and only if these pieces induce the same behavior under all program contexts. While the denotational semantics presented in [**?**] has been considered a definitive solution, we observe that Brookes's full abstraction result crucially relies on the availability of an impractical global atomic read-write action. We propose an alternative semantics that is based on traces that track program write actions together with the writes expected from the interfering environment, and is equipped by several closure operators to achieve necessary abstraction. We establish the adequacy of the semantics, and demonstrate full abstraction for a language with an atomic global read instruction or, in the absence of any global actions, for the case that the analyzed code segment is loop-free. To gain confidence, our results are mechanized in the Coq proof assistant.

1 INTRODUCTION

Denotational semantics aims to define the meaning of a piece of code independently of the context under which it is executed. Generally speaking, such semantics assigns a denotation $[\![C]\!]$ to every command C of a given programming language in a way that satisfies the following desiderata:

Compositionality: The denotation of a command should be determined from the denotations of the command's immediate constituents. For instance, assuming a sequential composition operator, ";", we require that $[\![C_1 \,;\, C_2]\!]$ is a function of $[\![C_1]\!]$ and $[\![C_2]\!]$.

Adequacy: Assuming a given operational semantics, the denotations should only consider equivalent commands that operationally behave the same when plugged in an arbitrary program context. When denotations are partially ordered, we also want the semantics to admit a *directional* version of adequacy that targets contextual refinement under the operational semantics instead of contextual equivalence. For instance, assuming that denotations are sets, as is the case in our development, (directional) adequacy ensures that $[\![C_1]\!] \subseteq [\![C_2]\!]$ implies that for every program context $P[-]$, every behavior of $P[C_1]$ under the operational semantics is also a behavior of $P[C_2]$. This makes denotations beneficial in supporting modular reasoning about the operational semantics, which by itself is only able to capture complete closed programs. In particular, an adequate denotational semantics can be used for formally justifying local program transformations, as performed by optimizing compilers. Indeed, adopting contextual refinement as the correctness criteria of program transformations, adequacy allows one to derive the correctness of a local transformation $C_{\mathsf{src}} \rightsquigarrow C_{\mathsf{tgt}}$ from $[\![C_{\mathsf{tgt}}]\!] \subseteq [\![C_{\mathsf{src}}]\!]$.

Full abstraction: Ideally, it is desirable for a denotational semantics to equate *all* pairs of commands that are contextually equivalent under the given operational semantics. A directional version requires that $[\![C_1]\!] \not\subseteq [\![C_2]\!]$ implies that for some program context $P[-]$, some behavior of $P[C_1]$ under the operational semantics is not a behavior of $P[C_2]$. Conceptually, full abstraction, together with compositionality and adequacy, means that $[\![C]\!]$ is indeed a compositional counterpart of the given operational semantics. More practically, a fully abstract denotational semantics provides a *complete* reasoning principle for correctness of local program transformations.

In this paper, we consider concurrent programs that employ shared variables for inter-thread synchronization, governed by a non-deterministic scheduler that cannot be controlled by the program. For this domain, developing compositional, adequate, and fully abstract semantics is highly challenging. Indeed, the standard approach for (non-deterministic) sequential programs, which models programs as transformations from an initial state to a set of final states, fails to provide compositional semantics for parallelism, since the state transformation induced by a parallel

composition $C_1 \parallel C_2$ is not uniquely determined from those of C_1 and C_2. One needs more detailed structures to capture the behaviors of C_1 and C_2, but being too concrete risks full abstraction.

This problem was solved by **?** (see there also a discussion about earlier attempts). In Brookes's approach, the semantics $[\![C]\!]$ of a command C is given by a set of sequences of transitions from memory to memory, assuming arbitrary environment interference between transitions. For example, a sequence of the form $\langle s_1, s_1' \rangle, \langle s_2, s_2' \rangle$ consisting of two transitions represents the case that C did some sequences of steps to transform s_1 to s_1'; then the environment did some steps transforming s_1' to s_2; and then C continued its execution from s_2 and terminated in s_2'. Brookes showed how these sequences can be derived from a given command by first deriving a concrete set of sequences, and then closing it under two closure operators, called *mumble* and *stutter*. In particular, $[\![C_1 \parallel C_2]\!]$ is obtained by considering all interleavings of sequences of $[\![C_1]\!]$ with sequences of $[\![C_2]\!]$, and closing the resulting set under the two closure operators. Brookes demonstrated compositionality, adequacy, and full abstraction for this semantics.

However, the programming language assumed in [**?**] employs a command of the form (`await` B `then` C) that implements a "conditional critical region": it blocks the execution as long as B is unmet, and then in a single atomic step it verifies that B holds and runs C. Since B and C may involve multiple variables, this construct can implement arbitrary atomic (finite) memory-to-memory transformations (e.g., $[\mathtt{x}_1 \mapsto 0, \ldots, \mathtt{x}_{100} \mapsto 0]$ to $[\mathtt{x}_1 \mapsto 1, \ldots, \mathtt{x}_{100} \mapsto 100]$), which requires all other components to be suspended and is unrealistic in practical concurrency. Removing (or restricting) `await` does not harm compositionality or adequacy, but, Brookes's full-abstraction proof heavily uses `await` instructions for building a concurrent context $P[-]$ that precisely mimics the environment transitions in a given sequence. In fact, the starting point for the current work is our observation that there are commands C_1 and C_2 that behave the same when plugged in program contexts without `await`, but can be differentiated by contexts that use `await` (see Examples 3.6, 4.6 and 5.1 below). Therefore, Brookes's semantics is too concrete for a language without `await`.

The main contribution of this work is to develop a novel denotational semantics that addresses this problem. We propose two models:

- A "concrete semantics" in which denotations track the write operations performed by the command interleaved with environment writes. For example, $\mathtt{W}(\mathtt{x}, 1), \bar{\mathtt{W}}(\mathtt{x}, 2), \mathtt{W}(\mathtt{y}, 1)$ represents the case that C writes 1 to x, expects the environment to write 2 to x, and then writes 1 to y. The concrete semantics is compositional and adequate, but it is not fully abstract. Nevertheless, especially since we do not record read operations in our denotations, the concrete semantics suffices for validating a wide variety of contextual refinements (see Fig. 4 below).
- An "abstract semantics" obtained by closing the concrete denotation under four rewrite rules, each of which mimics a certain operational simulation argument allowing one to hide and introduce component writes from the concrete trace. We show that the abstract semantics is also compositional and adequate, whereas full abstraction holds up to some level:
  - Full abstraction holds assuming a "snapshot" instruction that blocks the execution until some condition is met. This instruction is a restriction of `await` to instances of the form `await` B `then skip`.
  - Without "snapshot" we establish a restricted version of full abstraction: if C_2 is loop-free and $[\![C_1]\!] \not\subseteq [\![C_2]\!]$, then there exists a context $P[-]$ such that some behavior of $P[C_1]$ is not a behavior of $P[C_2]$. Thus, the abstract semantics is always *sound* for validating local program transformations $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$, and it provides a *complete* reasoning principle when C_{src} is loop-free. When C_{src} has loops, we provide (rather complicated) counterexample for full abstraction of our abstract semantics (see Example 4.28 below).

Instead of `await` instructions the language we assume employs standard read-modify-write (RMW) constructs that perform an atomic update of a *single* variable at a time. A natural question is whether, like `await`, RMWs allow concurrent contexts to distinguish between commands that are indistinguishable for contexts consisting solely of reads and writes. We answer this question affirmatively by demonstrating such cases (see Example 5.1 below). Moreover, we show that by strengthening one of the four rewrite rules used to define the abstract semantics, we obtain a denotational semantics that enables compositional reasoning about program transformations under the assumption that the context cannot perform RMW operations.

Finally, we note certain limitations of the current work (see also related work for further discussion). All of them raise interesting questions for future work, where our approach may constitute a solid starting point.

- We assume that the underlying memory ensures *sequential consistency* (SC, for short) [**?**]—the strongest memory model with simple operational semantics based on interleaving concurrent manipulations of a standard variables-to-values mapping.
- Our notion of behavior under the operational semantics is based on *partial correctness*, that is: we only consider terminating executions as inducing program behaviors. Accordingly, contextual refinement ensures that the target program preserves safety properties of the source, but it is termination-insensitive, where a diverging program refines every program. Since a compositional characterization of partial correctness is already challenging, we left the question of termination to future work. This is in line with multiple previous works that consider only terminating executions [**???**]. Nevertheless, **?** includes an extension to termination-sensitive refinement, using infinite sequences and assuming certain fairness assumptions on the operational semantics.
- Our programming language is a first-order language. Fully abstract semantics for higher-order languages have proved elusive [**?**], but we hope that our model can be useful for a higher-order language with a full abstraction guarantee that applies for its first-order fragment.

Outline. The rest of this paper is structured as follows. In §2, we present the syntax and operational semantics of the language studied in this paper. In §3, we present the concrete denotational semantics, establish its compositionality and adequacy, and demonstrate various transformations it validates (Fig. 4). In §4, we present the abstract denotational semantics, establish its compositionality (§4.1), adequacy (§4.2), and (restricted as discussed above) full abstraction (§4.3 and §4.4), and demonstrate transformations validated by the abstract semantics but not by the concrete one (Fig. 5). In §5, we present the modification of the abstract semantics under the assumption that the context does not perform RMWs. Finally, in §6, we discuss related and future work.

Supplementary Material. Our results are **mechanized in Coq**, and the mechanization is available in the supplementary material accompanying this submission. The only exceptions are the claim in Example 4.28 and Lemma 4.26 for which we provide written proofs in Appendix A.

2 SYNTAX, OPERATIONAL SEMANTICS, AND CONTEXTUAL REFINEMENT

In this section we present the syntax of the studied programming language, its operational semantics, and the notion of contextual refinement w.r.t. that semantics.

Syntax. We assume a set $\mathsf{Var} \triangleq \{\mathsf{x}, \mathsf{y}, \mathsf{z}, \ldots\}$ of shared variables, ranged over by x, y; a set $\mathsf{LVar} \triangleq \{\mathsf{a}, \mathsf{b}, \mathsf{c}, \ldots\}$ of local variables, ranged over by a, b; and a set $\mathsf{Val} \triangleq \{0, 1, 2, \ldots\}$ of values, ranged over by v. We define a *state* s to be a function in $\mathsf{State} \triangleq \mathsf{Var} \to \mathsf{Val}$. In some examples below, we use $\mathsf{s}_0 \triangleq \lambda x.\, 0$ as the initial state.

$$
\begin{array}{lll}
\text{expressions} & E ::= & a \mid v \mid E + E \mid E = E \mid \ \ldots \\
\text{let expressions} & L ::= & E \mid x \mid \mathtt{XCHG}(x, E) \mid \mathtt{FAA}(x, E) \mid \mathtt{CAS}(x, E, E) \\
\text{commands} & C ::= & \mathtt{skip} \mid x := E \mid \mathtt{let}\ a = L\ \mathtt{in}\ C \mid \\
 & & C\,;C \mid C \parallel C \mid C \oplus C \mid \mathtt{if}\ E\ \mathtt{then}\ C\ \mathtt{else}\ C \mid \mathtt{while}\ x\ \mathtt{do}\ C \mid \\
 & & \mathtt{assume}(E) \mid \mathtt{snapshot}(s) \mid x := * \mid \mathtt{while}\ *\ \mathtt{do}\ C \\
\text{contexts} & P ::= & - \mid \mathtt{let}\ a = L\ \mathtt{in}\ P \mid P\,;C \mid C\,;P \mid P \parallel C \mid C \parallel P \mid C \oplus P \mid P \oplus C \mid \\
 & & \mathtt{if}\ E\ \mathtt{then}\ P\ \mathtt{else}\ C \mid \mathtt{if}\ E\ \mathtt{then}\ C\ \mathtt{else}\ P \mid \mathtt{while}\ x\ \mathtt{do}\ P \mid \mathtt{while}\ *\ \mathtt{do}\ P
\end{array}
$$

Fig. 1. Syntax: Expressions, Let Expressions, Commands, and Contexts

Figure 1 presents the grammar for expressions, let expressions, commands, and contexts. Expressions are defined standardly and are composed of values and local variables. Let expressions are used on the right-hand side of let bindings, and include standard expressions, shared variables, and RMW primitives. The latter are used to atomically execute a read from memory followed by a write to memory. We consider three kinds of RMWs, whose intuitive semantics is as follows:

- Exchange (XCHG) loads from a shared variable and modifies it to a given argument.
- Fetch-And-Add (FAA) increments a shared variable by a given argument.
- Compare-And-Swap (CAS) compares a shared variable to a given argument, and, if the two values are equal, replaces the stored value by the value of the other argument.

All these instructions return the value they read, before the modification was performed.

Commands are mostly customary for a (first-order) imperative parallel language, with several choices that may deserve attention:

- Parallel composition, "$\parallel$", is a first class construct that can be employed deeply inside other commands, rather than top-level parallel composition which is sometimes assumed when studying semantics of parallel languages.
- We include non-deterministic choices—between commands ($C_1 \oplus C_2$), stored values ($x := *$), and as a loop termination condition ($\mathtt{while}\ *\ \mathtt{do}\ C$).
- Less standardly, we use functional-style "let" bindings for assigning values to local variables. This allows us to keep the scope of such variables indeed local to a given command, in a way that the parallel context cannot change or directly observe. Accordingly, while loops use a global variable for the termination condition.
- A non-standard $\mathtt{snapshot}(s)$ command is used to block the execution until the memory is in state s (see operational semantics below). As we discuss in §1, one of our full abstraction results depends on the availability of this command.

In examples, we also use ($\mathtt{if}\ E\ \mathtt{then}\ C$) for ($\mathtt{if}\ E\ \mathtt{then}\ C\ \mathtt{else}\ \mathtt{skip}$), and employ syntactic sugar incorporating loads/RMWs inside expressions, such as:

$$
\begin{array}{rcl}
x := y & \text{for} & \mathtt{let\ a} = y\ \mathtt{in}\ x := \mathtt{a} \\
\mathtt{if\ CAS}(x, v, v') = v\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 & \text{for} & \mathtt{let\ a} = \mathtt{CAS}(x, v, v')\ \mathtt{in\ (if\ a} = v\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2) \\
\mathtt{assume}(x = v) & \text{for} & \mathtt{let\ a} = x\ \mathtt{in\ assume(a} = v)
\end{array}
$$

We also denote by $\mathsf{fv}(E)$ (respectively, $\mathsf{fv}(C)$) the set of local variables that occur free in an expression E (command C), and call an expression E (command C) *closed* if $\mathsf{fv}(E) = \emptyset$ ($\mathsf{fv}(C) = \emptyset$). We write $C\{v/a\}$ for the command obtained from C by substituting the free occurrences of a by v.

Finally, Fig. 1 specifies "contexts" which are defined standardly as commands with one "hole". We write $P[C]$ for the command obtained by "plugging in" the command C in P, that is: substituting the unique $-$ in P by C.

$$\frac{v = [\![E]\!]}{\langle E, s\rangle \xrightarrow{\varepsilon} \langle v, s\rangle} \qquad \frac{v = s(x)}{\langle x, s\rangle \xrightarrow{\varepsilon} \langle v, s\rangle} \qquad \frac{v = s(x) \quad v' = [\![E]\!] \quad s' = s[x \mapsto v']}{\langle \mathtt{XCHG}(x, E), s\rangle \xrightarrow{\mathtt{W}(x,v')} \langle v, s'\rangle}$$

$$\frac{v = s(x) \quad v' = v + [\![E]\!] \quad s' = s[x \mapsto v']}{\langle \mathtt{FAA}(x, E), s\rangle \xrightarrow{\mathtt{W}(x,v')} \langle v, s'\rangle} \qquad \frac{v = s(x) = [\![E]\!] \quad v' = [\![E']\!] \quad s' = s[x \mapsto v']}{\langle \mathtt{CAS}(x, E, E'), s\rangle \xrightarrow{\mathtt{W}(x,v')} \langle v, s'\rangle} \qquad \frac{v = s(x) \neq [\![E]\!]}{\langle \mathtt{CAS}(x, E, \_), s\rangle \xrightarrow{\varepsilon} \langle v, s\rangle}$$

$$\frac{v = [\![E]\!] \quad s' = s[x \mapsto v]}{\langle x := E, s\rangle \xrightarrow{\mathtt{W}(x,v)} \langle \mathtt{skip}, s'\rangle} \qquad \frac{\langle L, s\rangle \xrightarrow{\gamma} \langle v, s'\rangle}{\langle \mathtt{let}\ a = L\ \mathtt{in}\ C, s\rangle \xrightarrow{\gamma} \langle C\{v/a\}, s'\rangle} \qquad \frac{\langle C_1, s\rangle \xrightarrow{\gamma} \langle C_1', s'\rangle}{\langle C_1 ; C_2, s\rangle \xrightarrow{\gamma} \langle C_1' ; C_2, s'\rangle}$$

$$\frac{}{\langle \mathtt{skip} ; C, s\rangle \xrightarrow{\varepsilon} \langle C, s\rangle} \qquad \frac{i \in \{1, 2\}}{\langle C_1 \oplus C_2, s\rangle \xrightarrow{\varepsilon} \langle C_i, s\rangle} \qquad \frac{\langle C_1, s\rangle \xrightarrow{\gamma} \langle C_1', s'\rangle}{\langle C_1 \parallel C_2, s\rangle \xrightarrow{\gamma} \langle C_1' \parallel C_2, s'\rangle}$$

$$\frac{\langle C_2, s\rangle \xrightarrow{\gamma} \langle C_2', s'\rangle}{\langle C_1 \parallel C_2, s\rangle \xrightarrow{\gamma} \langle C_1 \parallel C_2', s'\rangle} \qquad \frac{}{\langle \mathtt{skip} \parallel \mathtt{skip}, s\rangle \xrightarrow{\varepsilon} \langle \mathtt{skip}, s\rangle} \qquad \frac{[\![E]\!] \neq 0 \implies i = 1 \qquad [\![E]\!] = 0 \implies i = 2}{\langle \mathtt{if}\ E\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2, s\rangle \xrightarrow{\varepsilon} \langle C_i, s\rangle}$$

$$\frac{s(x) \neq 0}{\langle \mathtt{while}\ x\ \mathtt{do}\ C, s\rangle \xrightarrow{\varepsilon} \langle C ; \mathtt{while}\ x\ \mathtt{do}\ C, s\rangle} \qquad \frac{s(x) = 0}{\langle \mathtt{while}\ x\ \mathtt{do}\ C, s\rangle \xrightarrow{\varepsilon} \langle \mathtt{skip}, s\rangle} \qquad \frac{[\![E]\!] \neq 0}{\langle \mathtt{assume}(E), s\rangle \xrightarrow{\varepsilon} \langle \mathtt{skip}, s\rangle}$$

$$\frac{}{\langle \mathtt{snapshot}(s), s\rangle \xrightarrow{\varepsilon} \langle \mathtt{skip}, s\rangle} \qquad \frac{s' = s[x \mapsto v]}{\langle x := *, s\rangle \xrightarrow{\mathtt{W}(x,v)} \langle \mathtt{skip}, s'\rangle} \qquad \frac{}{\langle \mathtt{while}\ *\ \mathtt{do}\ C, s\rangle \xrightarrow{\varepsilon} \langle \mathtt{skip} \oplus (C ; \mathtt{while}\ *\ \mathtt{do}\ C), s\rangle}$$

Fig. 2. Small-Step Semantics: $\langle L, s\rangle \xrightarrow{\gamma} \langle v, s'\rangle$ and $\langle C, s\rangle \xrightarrow{\gamma} \langle C', s'\rangle$

Operational Semantics. We assume that closed expressions are evaluated to values using a function $[\![\cdot]\!]$ in a standard way. The operational semantics of commands is given in Fig. 2 as a "small-step" transition relation between configurations, which are tuples of the form $\langle C, s\rangle$, where C is a command and $s \in \mathrm{State}$. For a uniform definition of let bindings, it uses a "helper" relation which defines how let expressions are evaluated to values and affect the state.

The operational semantics is mostly standard. We use syntactic substitution to handle "let" bindings, so that steps assume the given command is closed. Parallelism follows non-preemptive scheduling allowing non-deterministic interleaving of component steps. The shared memory follows the SC model, where each read reads the latest written value recorded in the shared state. To assist later definitions, our small-step transitions are labeled with a write label of the form $\mathtt{W}(x, v)$ (with $x \in \mathrm{Var}$ and $v \in \mathrm{Val}$) or with an ε label when no write is performed. We often omit the label from the transition (i.e., $\langle C, s\rangle \to \langle C', s'\rangle$ means that $\langle C, s\rangle \xrightarrow{\gamma} \langle C', s'\rangle$ for some γ).

Note that no steps are associated with `skip` or with $\mathtt{assume}(E)$ when $[\![E]\!] = 0$. Intuitively, a state of the form $\langle \mathtt{skip}, s\rangle$ is a valid final state ("a value"). We write $\langle C, s\rangle \downarrow s'$ if $\langle C, s\rangle \to^* \langle \mathtt{skip}, s'\rangle$.[1]

Contextual Refinement. Contextual refinement under the operational semantics is identified with soundness of local program transformations, which is defined as follows:

[1]We denote by $S^?$ and S^* the reflexive and reflexive-transitive closures of a binary relation S (respectively) and write $S_1 ; S_2$ for relational composition, i.e., $S_1 ; S_2 \triangleq \{\langle x, z\rangle \mid \exists y.\, \langle x, y\rangle \in S_1 \wedge \langle y, z\rangle \in S_2\}$.

$$\frac{e \in \mathsf{EnvChro}}{\langle s, \theta, e\rangle \in \lfloor \mathtt{skip} \rfloor} \qquad \frac{e_1, e_2 \in \mathsf{EnvChro} \quad \alpha = \mathtt{W}(x, \theta(E))}{\langle s, \theta, e_1 \cdot \alpha \cdot e_2\rangle \in \lfloor x := E \rfloor} \qquad \frac{e \in \mathsf{EnvChro} \quad \langle \theta(L), e(s)\rangle \xrightarrow{\gamma} \langle v, s'\rangle \quad \langle s', \theta[a \mapsto v], c\rangle \in \lfloor C \rfloor}{\langle s, \theta, e \cdot \gamma \cdot c\rangle \in \lfloor \mathtt{let}\ a = L\ \mathtt{in}\ C \rfloor}$$

$$\frac{t_1 \in \lfloor C_1 \rfloor \quad t_2 \in \lfloor C_2 \rfloor}{t_1 \,;\, t_2 \in \lfloor C_1 \,;\, C_2 \rfloor} \qquad \frac{t_1 \in \lfloor C_1 \rfloor \quad t_2 \in \lfloor C_2 \rfloor}{t_1 \parallel t_2 \in \lfloor C_1 \parallel C_2 \rfloor} \qquad \frac{t \in \lfloor C_1 \rfloor \cup \lfloor C_2 \rfloor}{t \in \lfloor C_1 \oplus C_2 \rfloor} \qquad \frac{\theta(E) \neq 0 \implies \langle s, \theta, c\rangle \in \lfloor C_1 \rfloor \quad \theta(E) = 0 \implies \langle s, \theta, c\rangle \in \lfloor C_2 \rfloor}{\langle s, \theta, c\rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rfloor}$$

$$\frac{\langle e(s), \theta, c\rangle \in \lfloor C \rfloor \quad e(s)(x) \neq 0 \quad t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor}{\langle s, \theta, e \cdot c\rangle \,;\, t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor} \qquad \frac{e_1, e_2 \in \mathsf{EnvChro} \quad e_1(s)(x) = 0}{\langle s, \theta, e_1 \cdot e_2\rangle \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor} \qquad \frac{e \in \mathsf{EnvChro} \quad \theta(E) \neq 0}{\langle s, \theta, e\rangle \in \lfloor \mathtt{assume}(E) \rfloor}$$

$$\frac{e_1, e_2 \in \mathsf{EnvChro} \quad e_1(s) = s'}{\langle s, \theta, e_1 \cdot e_2\rangle \in \lfloor \mathtt{snapshot}(s') \rfloor} \qquad \frac{e_1, e_2 \in \mathsf{EnvChro} \quad \alpha = \mathtt{W}(x, \_)}{\langle s, \theta, e_1 \cdot \alpha \cdot e_2\rangle \in \lfloor x := * \rfloor} \qquad \frac{t_1 \in \lfloor C \rfloor \quad t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor}{t_1 \,;\, t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor} \qquad \frac{e \in \mathsf{EnvChro}}{\langle s, \theta, e\rangle \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor}$$

Fig. 3. Concrete Trace Semantics: $t \in \lfloor C \rfloor$

Definition 2.1. A transformation from a command C_{src} to a command C_{tgt} is *sound*, denoted by $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$, if $\langle P[C_{\mathrm{tgt}}], s\rangle \downarrow s'$ implies $\langle P[C_{\mathrm{src}}], s\rangle \downarrow s'$ for every context P such that $P[C_{\mathrm{src}}]$ and $P[C_{\mathrm{tgt}}]$ are closed. We write $C_1 \leftrightsquigarrow C_2$ when both $C_1 \rightsquigarrow C_2$ and $C_2 \rightsquigarrow C_1$ hold.

Example 2.2. For $C_1 = \mathtt{x} := \mathtt{x} + 1$ and $C_2 = \mathtt{let\ a} = \mathtt{FAA(x, 1)\ in\ skip}$, we have $C_1 \rightsquigarrow C_2$ but $C_2 \not\rightsquigarrow C_1$. For the former, we can execute a load followed by a store in one atomic step to simulate the effect of FAA. (The denotational semantics below provides a formal account.) For the latter, with $P = - \parallel \mathtt{x} := 1 \,;\, \mathtt{x} := 3$, we have $\langle P[C_i], \mathtt{s}_0\rangle \downarrow \mathtt{s}_0[\mathtt{x} \mapsto 2]$ for $i = 1$ but not for $i = 2$.

3 CONCRETE DENOTATIONAL SEMANTICS

In this section we present the "concrete" denotational semantics and establish its compositionality and adequacy. The main ingredient for this semantics is our notion of a *trace*, which consists of an initial memory state, an initial *store*, which assigns values to local variables, and a *chronicle*, which is a sequence of *actions* performed by the command along with those expected by the concurrent context. Next, we formally define these objects.

Notation 3.1 (Sequences). For a finite alphabet Σ, we denote by Σ^* the set of all (finite) sequences over Σ. We use ε to denote the empty sequence. We write $s_1 \cdot s_2$ for the concatenation of sequences, which is lifted to concatenation of sets of sequences in the obvious way. We identify symbols with sequences of length 1 or their singletons when needed (e.g., in expressions like $\sigma \cdot S$).

Stores. A *store* is a function $\theta \in \mathsf{Store} \triangleq \mathsf{LVar} \to \mathsf{Val}$. Stores are extended to expressions and let expressions in the standard way. In some examples below, we use $\theta_0 \triangleq \lambda a.\, 0$ as the initial store.

Actions. An *action* α is either a *component write* of the form $\mathtt{W}(x, v)$ with $x \in \mathsf{Var}$ and $v \in \mathsf{Val}$, or an *environment write* of the form $\bar{\mathtt{W}}(x, v)$ with $x \in \mathsf{Var}$ and $v \in \mathsf{Val}$. We write Act, CmpW, and EnvW for the set of all actions, component writes, and environment writes (respectively).

Chronicles. A *chronicle* c is a finite sequence of actions. We denote by Chro the set of all chronicles, by CmpChro the set of all chronicles consisting solely of component writes, and by EnvChro the

set of all chronicles consisting solely of environment writes. A chronicle c induces a function from states to states, recursively defined by: $\varepsilon(s) \triangleq s$ and $(\mathtt{W}(x,v)\cdot c)(s) = (\bar{\mathtt{W}}(x,v)\cdot c)(s) \triangleq c(s[x \mapsto v])$.

Traces. A *trace* is a triple $t = \langle s, \theta, c\rangle \in \mathsf{Trace} \triangleq \mathsf{State} \times \mathsf{Store} \times \mathsf{Chro}$. We refer to the three components as the initial state (s), the initial store (θ), and the chronicle (c) of t, and to the state $c(s)$ as the *final state* of the trace t.

From Commands to Traces. The concrete semantics is given as a function $\lfloor\cdot\rfloor$ which maps commands to sets of traces. An inductive definition is given in Fig. 3.

The `skip` command does not perform any component writes and tolerates any environment interference, thus it is associated with traces with arbitrary environment chronicles. A store instruction generates a component write, and allows arbitrary environment interference before and after. The value to be stored is determined according to the initial store. (Unlike the operational semantics, this semantics assigns meaning to open programs as well.)

Let bindings start with environment interference e and then possibly generate a component write $\mathtt{W}(x,v)$ following their operational semantics (reusing the first part of Fig. 2). Note that the memory visible to the let expression is the one obtained by applying e on the initial state. In turn, the continuation is given by C starting from a modified state and store. The resulting chronicle is the concatenation of e, $\gamma \in \{\mathtt{W}(x,v), \varepsilon\}$, and a chronicle c of the continuation. Here we use the transition labels from the operational semantics as component actions or the empty chronicles.

Sequential composition of commands is handled by sequential composition of traces. The sequential composition of $t = \langle s, \theta, c\rangle$ and $t' = \langle c(s), \theta, c'\rangle$, denoted by $t\,;t'$, is the trace $\langle s, \theta, c\cdot c'\rangle$. When the final state of t does not coincide with the initial state of t' or the two traces do not have the same initial store, then $t\,;t'$ is undefined.

The denotation of parallel composition uses a (partial) operation for parallel composition of traces. Intuitively speaking, a component action on one side has to match the environment action expected from the other side, and together they form a component action for their external environment. In addition, if both sides expect the same environment action, then that action is also expected from the external environment of the parallel composition. Formally, parallel composition of traces is built on top of parallel composition of actions and of chronicles:

(1) The *dual* of an action α, denoted by $\bar{\alpha}$, is defined by $\bar{\alpha} \triangleq \bar{\mathtt{W}}(x,v)$ if $\alpha = \mathtt{W}(x,v)$ and $\bar{\alpha} \triangleq \mathtt{W}(x,v)$ if $\alpha = \bar{\mathtt{W}}(x,v)$. Two actions α and α' are *parallelly composable* if either $\alpha = \bar{\alpha'}$ or $\alpha = \alpha' \in \mathsf{EnvW}$. In that case, their *parallel composition*, denoted by $\alpha \parallel \alpha'$, is given by:

$$\alpha \parallel \alpha' \triangleq \begin{cases} \mathtt{W}(x,v) & \alpha = \bar{\alpha'} \in \{\mathtt{W}(x,v), \bar{\mathtt{W}}(x,v)\} \\ \alpha & \alpha = \alpha' \in \mathsf{EnvW} \end{cases}$$

(2) The *parallel composition* of two chronicles $c = \alpha_1 \cdots \alpha_n$ and $c' = \alpha'_1 \cdots \alpha'_n$, denoted by $c_1 \parallel c_2$, is defined by $c \parallel c' \triangleq (\alpha_1 \parallel \alpha'_1) \cdots (\alpha_n \parallel \alpha'_n)$. If some $\alpha_i \parallel \alpha'_i$ is undefined or the chronicles are not of the same length, then $c \parallel c'$ is undefined.

(3) The *parallel composition* of two traces $t = \langle s, \theta, c\rangle$ and $t' = \langle s, \theta, c'\rangle$, denoted by $t \parallel t'$, is the trace $\langle s, \theta, c \parallel c'\rangle$. When the two traces do not have the same initial state and store or $c \parallel c'$ is undefined, then $t \parallel t'$ is undefined.

The concrete semantics of other language constructs follow similar ideas aiming to match their operational semantics. As expected, for loops, the definition is recursive.

Example 3.2. For $C = \texttt{let a = x in (x := a + 1)}$, $\lfloor C\rfloor$ consists of all traces of the form $\langle s, \theta, e_1 \cdot e_2 \cdot \mathtt{W(x}, v+1) \cdot e_3\rangle$ where $e_1, e_2, e_3 \in \mathsf{EnvChro}$ and $v = e_1(s)(\mathtt{x})$. In addition, $\lfloor \mathtt{x := 1}\rfloor$ consists of all traces of the form $\langle s, \theta, e_1 \cdot \mathtt{W(x}, 1) \cdot e_2\rangle$ where $e_1, e_2 \in \mathsf{EnvChro}$. For their parallel composition, $\lfloor C \parallel \mathtt{x := 1}\rfloor$ consists of all traces of the form $\langle s, \theta, e_1 \cdot e_2 \cdot \mathtt{W(x}, v+1) \cdot e_3 \cdot \mathtt{W(x}, 1) \cdot e_4\rangle$

or $\langle s, \theta, e_1 \cdot e_2 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_3 \cdot \mathtt{W}(\mathsf{x}, v+1) \cdot e_4 \rangle$ where $e_1, e_2, e_3, e_4 \in \mathsf{EnvChro}$ and $v = e_1(s)(\mathsf{x})$; and $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \cdot e_3 \cdot \mathtt{W}(\mathsf{x}, v+1) \cdot e_4 \rangle$ where $e_1, e_2, e_3, e_4 \in \mathsf{EnvChro}$ and $v = e_2(s[\mathsf{x} \mapsto 1])(\mathsf{x})$.

Compositionality. From the definition of the semantics, it is easy to see that the concrete semantics is compositional. More formally, the following property is proved by standard induction on contexts (with an inner induction on the derivation of $t \in \lfloor \mathtt{while}\ \_\ \mathtt{do}\ C \rfloor$ for loops):

Lemma 3.3. *If $\lfloor C_1 \rfloor \subseteq \lfloor C_2 \rfloor$, then $\lfloor P[C_1] \rfloor \subseteq \lfloor P[C_2] \rfloor$ for every context P.*

As a corollary, we obtain the compositionality of $\lfloor \cdot \rfloor$: for every command C whose immediate sub-commands are $C_1, \ldots, C_n$, we have that $\lfloor C \rfloor$ is a function of $\lfloor C_1 \rfloor, \ldots, \lfloor C_n \rfloor$. To see this, consider for instance the case of $C = C_1 \parallel C_2$, and suppose that $\lfloor C_1 \rfloor = \lfloor C_1' \rfloor$ and $\lfloor C_2 \rfloor = \lfloor C_2' \rfloor$. By Lemma 3.3 applied to the context $P = - \parallel C_2$ and the commands C_1, C_1', we obtain $\lfloor C_1 \parallel C_2 \rfloor = \lfloor C_1' \parallel C_2 \rfloor$. Then, again by Lemma 3.3 applied to the context $P = C_1' \parallel -$ and the commands C_2, C_2', we obtain $\lfloor C_1' \parallel C_2 \rfloor = \lfloor C_1' \parallel C_2' \rfloor$. Together, it follows that $\lfloor C_1 \parallel C_2 \rfloor = \lfloor C_1' \parallel C_2' \rfloor$.

Adequacy. The next lemma provides the key for the adequacy of the concrete semantics. Its proof employs *labeled interrupted executions* (akin to ?'s "state trace" behaviors).

Lemma 3.4. *For a closed command C, we have $\langle C, s \rangle \downarrow s'$ iff $\langle s, \theta, c \rangle \in \lfloor C \rfloor$ for some store θ and component chronicle c such that $c(s) = s'$.*

Proof. For the proof we inductively define an auxiliary relation $\overset{c}{\Rightarrow}$ between configurations labeled with a chronicle c, which represents an operational execution interrupted with the environment writes along c:

$$\frac{}{\langle C, s \rangle \overset{\varepsilon}{\Rightarrow} \langle C, s \rangle} \qquad \frac{\alpha = \bar{\mathtt{W}}(x, v) \quad \langle C, s[x \mapsto v] \rangle \overset{c}{\Rightarrow} \langle C', s' \rangle}{\langle C, s \rangle \overset{\alpha \cdot c}{\Longrightarrow} \langle C', s' \rangle} \qquad \frac{\langle C, s \rangle \xrightarrow{\gamma} \langle C', s' \rangle \quad \langle C', s' \rangle \overset{c}{\Rightarrow} \langle C'', s'' \rangle}{\langle C, s \rangle \overset{\gamma \cdot c}{\Longrightarrow} \langle C'', s'' \rangle}$$

When c is a component chronicle, $\overset{c}{\Rightarrow}$ trivially coincides with the operational semantics:

Claim 3.4.1: $\langle C, s \rangle \downarrow s'$ iff $\langle C, s \rangle \overset{c}{\Rightarrow} \langle \mathtt{skip}, s' \rangle$ for some $c \in \mathsf{CmpChro}$ such that $c(s) = s'$.

In addition, by induction on C, we can establish the correspondence between $\lfloor C \rfloor$ and $\overset{c}{\Rightarrow}$:

Claim 3.4.2: Let C be a command, and let $a_1, \ldots, a_n$ be an enumeration of $\mathsf{fv}(C)$. Then, $\langle s, \theta, c \rangle \in \lfloor C \rfloor$ iff $\langle C\{\theta(a_1)/a_1\} \ldots \{\theta(a_n)/a_n\}, s \rangle \overset{c}{\Rightarrow} \langle \mathtt{skip}, c(s) \rangle$.

The claim in Lemma 3.4 is a direct corollary of these two claims. □

Adequacy of the concrete semantics is now a corollary:

Theorem 3.5. *If $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq \lfloor C_{\mathrm{src}} \rfloor$, then $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$.*

Proof. Suppose that $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq \lfloor C_{\mathrm{src}} \rfloor$. Let P be a context such that $P[C_{\mathrm{src}}]$ and $P[C_{\mathrm{tgt}}]$ are closed, and suppose that $\langle P[C_{\mathrm{tgt}}], s \rangle \downarrow s'$. Since $\langle P[C_{\mathrm{tgt}}], s \rangle \downarrow s'$, by Lemma 3.4, we have $\langle s, \theta, c \rangle \in \lfloor P[C_{\mathrm{tgt}}] \rfloor$ for some store θ and component chronicle $c \in \mathsf{CmpChro}$ such that $c(s) = s'$. Since $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq \lfloor C_{\mathrm{src}} \rfloor$, by Lemma 3.3, it follows that $\langle s, \theta, c \rangle \in \lfloor P[C_{\mathrm{src}}] \rfloor$. By Lemma 3.4, it follows that $\langle P[C_{\mathrm{src}}], s \rangle \downarrow s'$. □

Figure 4 presents examples of program transformations that are validated by the concrete semantics. Among RMWs, we only list transformations involving FAA, but similar transformations can be shown for XCHG and CAS. As a concrete example of the style of reasoning in these proofs, consider the case unused load elimination/introduction. The traces in $\lfloor \mathtt{let}\ a = x\ \mathtt{in}\ C \rfloor$ are by

Algebraic laws of sequential composition

$(C_1 \,;\, C_2) \,;\, C_3 \leftrightsquigarrow C_1 \,;\, (C_2 \,;\, C_3) \qquad \texttt{skip} \,;\, C \leftrightsquigarrow C \qquad C \,;\, \texttt{skip} \leftrightsquigarrow C$

Reordering of local operations

$$\begin{array}{l}\texttt{let } a = E \texttt{ in} \\ \texttt{let } a' = E' \texttt{ in } C\end{array} \leftrightsquigarrow \begin{array}{l}\texttt{let } a' = E' \texttt{ in} \\ \texttt{let } a = E \texttt{ in } C\end{array} \quad \text{provided that } a \neq a',\ a \notin \mathrm{fv}(E'),\ \text{and } a' \notin \mathrm{fv}(E)$$

$$\texttt{let } a = E \texttt{ in } x := E \,;\, C \leftrightsquigarrow \texttt{let } a = E \texttt{ in } x := a \,;\, C \quad \text{provided that } a \notin \mathrm{fv}(E)$$

$$\begin{array}{l}\texttt{if } E \texttt{ then } (\texttt{let } a = E' \texttt{ in } C_1) \\ \texttt{else } (\texttt{let } a = E' \texttt{ in } C_2)\end{array} \leftrightsquigarrow \begin{array}{l}\texttt{let } a = E' \texttt{ in} \\ (\texttt{if } E \texttt{ then } C_1 \texttt{ else } C_2)\end{array} \quad \text{provided that } a \notin \mathrm{fv}(E)$$

Unused assignment elimination

$\texttt{let } a = E \texttt{ in } C \leftrightsquigarrow C$ provided that $a \notin \mathrm{fv}(C)$

Loop unrolling

$\texttt{while } x \texttt{ do } C \leftrightsquigarrow \texttt{let } a = x \texttt{ in } (\texttt{if } a \texttt{ then } (C \,;\, \texttt{while } x \texttt{ do } C) \texttt{ else skip})$ provided that $a \notin \mathrm{fv}(C)$

Algebraic laws of parallel composition

$\texttt{skip} \parallel C \leftrightsquigarrow C \qquad C_1 \parallel C_2 \leftrightsquigarrow C_2 \parallel C_1 \qquad (C_1 \parallel C_2) \parallel C_3 \leftrightsquigarrow C_1 \parallel (C_2 \parallel C_3)$

Generalized Sequencing (a.k.a. thread inlining/sequentialization)

$(C_1 \,;\, C_1') \parallel (C_2 \,;\, C_2') \rightsquigarrow (C_1 \parallel C_2) \,;\, (C_1' \parallel C_2')$

Algebraic laws of non-deterministic choice and distributivity over non-deterministic choice

$C_1 \oplus C_2 \leftrightsquigarrow C_2 \oplus C_1 \qquad (C_1 \oplus C_2) \oplus C_3 \leftrightsquigarrow C_1 \oplus (C_2 \oplus C_3) \qquad C \oplus C \leftrightsquigarrow C \qquad C_1 \oplus C_2 \rightsquigarrow C_1 \qquad C_1 \oplus C_2 \rightsquigarrow C_2$
$C \,;\, (C_1 \oplus C_2) \leftrightsquigarrow (C \,;\, C_1) \oplus (C \,;\, C_2) \qquad (C_1 \oplus C_2) \,;\, C \leftrightsquigarrow (C_1 \,;\, C) \oplus (C_2 \,;\, C)$
$C \parallel (C_1 \oplus C_2) \leftrightsquigarrow (C \parallel C_1) \oplus (C \parallel C_2)$

Load-after-load elimination

$\texttt{let } a = x \texttt{ in } (\texttt{let } b = x \texttt{ in } C) \rightsquigarrow \texttt{let } a = x \texttt{ in } (\texttt{let } b = a \texttt{ in } C)$

Load-after-store elimination

$x := E \,;\, \texttt{let } a = x \texttt{ in } C \rightsquigarrow x := E \,;\, \texttt{let } a = E \texttt{ in } C$

Unused load elimination/introduction

$\texttt{let } a = x \texttt{ in } C \leftrightsquigarrow C$ provided that $a \notin \mathrm{fv}(C)$

Load-after-FAA elimination

$$\begin{array}{l}\texttt{let } a = \texttt{FAA}(x, E) \texttt{ in} \\ \texttt{let } a' = x \texttt{ in } C\end{array} \rightsquigarrow \begin{array}{l}\texttt{let } a = \texttt{FAA}(x, E) \texttt{ in} \\ \texttt{let } a' = a + E \texttt{ in } C\end{array} \quad \text{provided that } a \notin \mathrm{fv}(E)$$

Load-before-FAA elimination

$$\begin{array}{l}\texttt{let } a = x \texttt{ in} \\ \texttt{let } a' = \texttt{FAA}(x, E) \texttt{ in } C\end{array} \rightsquigarrow \begin{array}{l}\texttt{let } a = \texttt{FAA}(x, E) \texttt{ in} \\ \texttt{let } a' = a \texttt{ in } C\end{array} \quad \text{provided that } a \notin \mathrm{fv}(E)$$

Load-Store to FAA

$\texttt{let } a = x \texttt{ in } (x := a + E \,;\, C) \rightsquigarrow \texttt{let } a = \texttt{FAA}(x, E) \texttt{ in } C$ provided that $a \notin \mathrm{fv}(E)$

Assume introduction

$\texttt{skip} \rightsquigarrow \texttt{assume}(E) \qquad \texttt{skip} \rightsquigarrow \texttt{let } a = x \texttt{ in } \texttt{assume}(E) \qquad C \rightsquigarrow \texttt{assume}(0)$

Refinement of non-determinism

$x := * \rightsquigarrow x := v \qquad \texttt{while } * \texttt{ do } C \rightsquigarrow \texttt{while } x \texttt{ do } C$

Fig. 4. Examples of program transformations validated by the concrete semantics

definition of the form $\langle s, \theta, e \cdot c\rangle$ with $e \in \mathsf{EnvChro}$ and $\langle e(s), \theta[a \mapsto e(s)(x)], c\rangle \in \lfloor C \rfloor$. When $a \notin \mathrm{fv}(C)$, the latter holds iff $\langle e(s), \theta, c\rangle \in \lfloor C \rfloor$. Then, by picking $e = \varepsilon$, we obtain $\lfloor C \rfloor \subseteq \lfloor \texttt{let } a = x \texttt{ in } C \rfloor$ (load elimination). The converse (load introduction) follows from the observation that $\langle e(s), \theta, c\rangle \in \lfloor C \rfloor$ implies that $\langle s, \theta, e \cdot c\rangle \in \lfloor C \rfloor$ for every command C. Our Coq development formalizes all examples and provides general lemmas that are repeatedly used in these arguments.

Example 3.6. The concrete semantics captures some refinements that are invalid in [?], which allows the concurrent context to form arbitrary atomic blocks. Indeed, every command in the language we study changes at most one shared variable. This is reflected in traces since every action in them mentions one variable. For instance, using the concrete semantics we can show that $C_1 \,;\, C_2 \,;\, C_3 \rightsquigarrow C_1 \,;\, C_3$ for:

$$C_1 = \mathtt{x := 1 \,;\, y := 1}$$
$$C_2 = \mathtt{let\ a = x\ in\ (let\ b = y\ in\ (assume((a = 2 \wedge b \neq 2) \vee (a \neq 2 \wedge b = 2))))}$$
$$C_3 = \mathtt{let\ a = x\ in\ (let\ b = y\ in\ (assume((a = 2 \wedge b = 2))))}$$

This refinement fails to hold if the context concurrently performs `await true then (x := 2 ; y := 2)`.

4 ABSTRACT SEMANTICS

The semantics above fully tracks the sequence of writes performed by a command. There are, however, contextual refinements in which writes are eliminated or introduced. The "abstract semantics" presented in this section supports such refinements. The main idea is to close the concrete sets of traces under certain rewrite rules that hide or introduce actions that can be safely assumed to be unobservable by the concurrent environment. Then, it may be the case that some traces in $\lfloor C_{\mathrm{tgt}} \rfloor$ are not in $\lfloor C_{\mathrm{src}} \rfloor$, but they are in the closure of $\lfloor C_{\mathrm{src}} \rfloor$ under these rewrites.

The main technical challenge lies in identifying these rewrite rules and proving the required properties for this semantics. First, in §4.1, we establish the compositionality property, which, unlike the case of the concrete semantics, is not a direct corollary of the definition, and requires a completely new kind of argument. Then, in §4.2, we show how adequacy of the abstract semantics follows from its compositionality and Lemma 3.4 about the concrete semantics. Finally, in §4.3, we show that the set of rules is "complete" by establishing full abstraction, and in §4.4 we consider the case that "snapshots" are not available.

Notation 4.1 (Rewrite Rules and Closures). A *rewrite rule* is a binary relation, where we use the notation $a \xrightarrow{x} b$ to mean that $\langle a, b\rangle \in x$. For a set X of rewrite rules, we write $a \xrightarrow{X} b$ if $a \xrightarrow{x} b$ for some $x \in X$. A set A is *closed* under X if $b \in A$ whenever $a \xrightarrow{X} b$ for some $a \in A$. Assuming some universal set $\mathcal{A}$, the *closure* of A under X, denoted by A^X, is defined as the smallest subset of $\mathcal{A}$ that contains A and is closed under X.

The following general propositions are useful in the sequel.

Proposition 4.2. *For every $A \subseteq \mathcal{A}$ and set X of rewrite rules, $A^X = \{b \in \mathcal{A} \mid \exists a \in A.\, a \xrightarrow{X}{}^{*}\ b\}$*

Proposition 4.3. *For every $A, B \subseteq \mathcal{A}$ and set X of rewrite rules, $A \subseteq B^X$ implies $A^X \subseteq B^X$.*

Now, we define the abstract semantics.

Definition 4.4. The *abstract denotation* of a command C, denoted by $[\![C]\!]$, is defined by $[\![C]\!] \triangleq \lfloor C \rfloor^{\mathcal{R}}$, where $\mathcal{R} \triangleq \{\xrightarrow{\mathrm{r}_1}, \xrightarrow{\mathrm{r}_2}, \xrightarrow{\mathrm{r}_3}, \xrightarrow{\mathrm{r}_4}\}$ consists of the following rewrite rules on traces:

(1) $\langle s, \theta, c_1 \cdot m_1 \cdot \mathtt{W}(x, v) \cdot m_2 \cdot c_2\rangle \xrightarrow{\mathrm{r}_1} \langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2\rangle$ provided that $m_1, m_2 \in \mathrm{CmpChro}$ and $(m_1 \cdot \mathtt{W}(x, v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$.
(2) $\langle s, \theta, c_1 \cdot m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2 \cdot c_2\rangle \xrightarrow{\mathrm{r}_2} \langle s, \theta, c_1 \cdot \bar{\mathtt{W}}(x, v) \cdot c_2\rangle$ provided that $m_1, m_2 \in \mathrm{CmpChro}$, $(m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$, and $m_1(c_1(s))(x) = c_1(s)(x)$.
(3) $\langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2\rangle \xrightarrow{\mathrm{r}_3} \langle s, \theta, c_1 \cdot c_2\rangle$ provided that $c_1(s)(x) = v$.
(4) $\langle s, \theta, c_1 \cdot c_2\rangle \xrightarrow{\mathrm{r}_4} \langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2\rangle$ provided that $c_1(s)(x) = v$.

Next, we demonstrate each of the rules in a concrete example, and show how these rules mimic intuitive operational reasoning in a way that is impossible in the concrete semantics. In these examples, we freely use Prop. 4.3 and show $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq [\![C_{\mathrm{src}}]\!]$ instead of $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$.

Example 4.5 (Rule r_1*).* Rule r_1 permits the combination of several component write that have no intervening environment write. The condition $(m_1 \cdot \mathtt{W}(x,v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$ ensures that the effect of the formed block is the same as the effect of a single store. As an example, let $C_{\mathrm{src}} =$ `let a = y in (y := 1 ; x := 1 ; y := a)`. Intuitively speaking, we can always execute it atomically, without letting the environment to interfere in between the first load and the final store. In that case, it behaves like $C_{\mathrm{tgt}} =$ `x := 1`. Such reasoning is impossible in the concrete semantics, but the rule r_1 of the abstract semantics is allowing us exactly that, thus justifying $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$. Indeed, to show that $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq [\![C_{\mathrm{src}}]\!]$ observe that every trace t in $\lfloor C_{\mathrm{tgt}} \rfloor$ has the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \rangle$. We can start from a corresponding trace in $[\![C_{\mathrm{src}}]\!]$ of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{y}, 1) \cdot \mathtt{W}(\mathsf{x}, 1) \cdot \mathtt{W}(\mathsf{y}, e_1(s)(\mathsf{y})) \cdot e_2 \rangle$ and rewrite by r_1 with $c_1 = e_1$, $m_1 = \mathtt{W}(\mathsf{y}, 1)$, $m_2 = \mathtt{W}(\mathsf{y}, e_1(s)(\mathsf{y}))$, and $c_2 = e_2$ to obtain t.

Example 4.6 (Rule r_2*).* Rule r_2 allows one to "attach" component actions to an environment action, provided that the composed block has the same effect as the single environment action. To see this in action, let $C_{\mathrm{src}} =$ `let a = y in y := 1 ; if x ≠ 0 then (if x = 0 then y := a)`. The two if conditions are satisfied only if the concurrent environment changes x from non-zero value to zero. In this case, we can encompass that environment store of x with the load from y and the store of 1 to y just before, and the store of the previous value of y just after, and in this case C_{src} behaves like $C_{\mathrm{tgt}} =$ `if x ≠ 0 then (if x = 0 then skip else y := 1) else y := 1`. The rule r_2 of the abstract semantics is needed for that, thus justifying $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$. Indeed, to show that $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq [\![C_{\mathrm{src}}]\!]$ observe that the traces in $\lfloor C_{\mathrm{tgt}} \rfloor$ are either of the form $\langle s, \theta, e_1 \cdot \bar{\mathtt{W}}(\mathsf{x}, 0) \cdot e_2 \rangle$ with $e_1(s)(\mathsf{x}) \neq 0$ or of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{y}, 1) \cdot e_2 \rangle$. Traces of the latter form are directly in $\lfloor C_{\mathrm{src}} \rfloor$. For a trace t of the first form, we start from a corresponding trace in $\lfloor C_{\mathrm{src}} \rfloor$ of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{y}, 1) \cdot \bar{\mathtt{W}}(\mathsf{x}, 0) \cdot \mathtt{W}(\mathsf{y}, e_1(s)(\mathsf{y})) \cdot e_2 \rangle$ and rewrite by r_2 with $c_1 = e_1$, $m_1 = \mathtt{W}(\mathsf{y}, 1)$, $m_2 = \mathtt{W}(\mathsf{y}, e_1(s)(\mathsf{y}))$, and $c_2 = e_2$ to obtain t. As in r_1, the condition $(m_1 \cdot \bar{\mathtt{W}}(x,v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$ of r_2 ensures that the formed atomic block affects the memory exactly as the single environment store.

Example 4.7 (Second side-condition of r_2*).* Attaching component actions to an environment write $\bar{\mathtt{W}}(x,v)$, may fail if the component actions modify x and the environment write is due to an RMW that depends on the value of x. This is the reason for the condition $m_1(c_1(s))(x) = c_1(s)(x)$ in r_2. For example, if we use x instead of y in Example 4.6, without this condition, we would obtain

`let a = x in x := 1 ; if x ≠ 0 then (if x = 0 then x := a)` $\rightsquigarrow$
`if x ≠ 0 then (if x = 0 then skip else x := 1) else x := 1`

which must not hold. Indeed, starting from a state with $\mathsf{x} \mapsto 2$, in parallel to $\mathtt{CAS}(\mathsf{x}, 2, 0)$, only the target program can terminate in a state in which $\mathsf{x} \mapsto 0$.

Example 4.8 (Rule r_3*).* Executing $C_{\mathrm{src}} =$ `let a = x in x := a` atomically is invisible for the concurrent enviorment, behaving like `skip`. In the concrete semantics, we cannot prove $C_{\mathrm{src}} \rightsquigarrow$ `skip` since all chronicles of $\lfloor C_{\mathrm{src}} \rfloor$ have one component write, whereas those of $\lfloor$`skip`$\rfloor$ have none. The rule r_3 of the abstract semantics is needed here. Indeed, to show that $\lfloor$`skip`$\rfloor \subseteq [\![C_{\mathrm{src}}]\!]$, we start with an arbitrary trace t in $\lfloor$`skip`$\rfloor$, which must have the form $\langle s, \theta, e \rangle$. Then, a corresponding trace in $\lfloor C_{\mathrm{src}} \rfloor$ of the form $\langle s, \theta, e \cdot \mathtt{W}(\mathsf{x}, e(s)(\mathsf{x})) \rangle$ can be rewritten to t by r_3 with $c_1 = e$ and $c_2 = \varepsilon$.

Example 4.9 (Rule r_4*).* In the operational semantics fetch-and-add by 0 is equivalent to a read. In particular, for $C_{\mathrm{src}} =$ `let a = x in y := a` and $C_{\mathrm{tgt}} =$ `let a =` $\mathtt{FAA}(\mathsf{x}, 0)$ `in y := a`, we have $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$. This cannot be shown by the concrete semantics since chronicles of C_{src} have only

one component write, while those of C_{tgt} have two such writes. To show that $\lfloor C_{\text{tgt}} \rfloor \subseteq [\![C_{\text{src}}]\!]$, we start with an arbitrary trace t in $\lfloor C_{\text{tgt}} \rfloor$, which must have the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, v) \cdot e_2 \cdot \mathtt{W}(\mathsf{y}, v) \cdot e_3 \rangle$ with $v = e_1(s)(\mathsf{x})$. Then, a corresponding trace in $\lfloor C_{\text{src}} \rfloor$ of the form $\langle s, \theta, e_1 \cdot e_2 \cdot \mathtt{W}(\mathsf{y}, v) \cdot e_3 \rangle$ can be rewritten to t by r_4 with $c_1 = e_1$ and $c_2 = e_2 \cdot \mathtt{W}(\mathsf{y}, v) \cdot e_3$.

Example 4.10. Rule r_4 is also necessary for a language without RMWs. For:

$$C_{\text{src}} = \mathtt{let\ a = y\ in\ (y := 1\ ;\ if\ x \neq 0\ then\ x := 0\ else\ skip\ ;\ y := a)}$$
$$C_{\text{tgt}} = \mathtt{assume(x = 0)\ ;\ x := 0}$$

we have $C_{\text{src}} \rightsquigarrow C_{\text{tgt}}$, but $\lfloor C_{\text{tgt}} \rfloor \subseteq [\![C_{\text{src}}]\!]$ cannot be established without r_4.

Remark 4.11. In the presence of r_1 and r_4, the rewrite rule r_3 can be strengthened as follows:

(3') $\langle s, \theta, c_1 \cdot m \cdot c_2 \rangle \xrightarrow{\mathsf{r}_3'} \langle s, \theta, c_1 \cdot c_2 \rangle$ provided that $m \in \mathsf{CmpChro}$ and $m(c_1(s)) = c_1(s)$.

Indeed, we can always pick an arbitrary variable $x \in \mathsf{Var}$ and rewrite as follows:

$$\langle s, \theta, c_1 \cdot m \cdot c_2 \rangle \xrightarrow{\mathsf{r}_4} \langle s, \theta, c_1 \cdot m \cdot \mathtt{W}(x, c_1(s)(x)) \cdot c_2 \rangle \xrightarrow{\mathsf{r}_1} \langle s, \theta, c_1 \cdot \mathtt{W}(x, c_1(s)(x)) \cdot c_2 \rangle \xrightarrow{\mathsf{r}_3} \langle s, \theta, c_1 \cdot c_2 \rangle.$$

4.1 Compositionality

Next, we establish the compositionality of $[\![C]\!]$. First, to handle sequential composition, we observe that the rules of $\mathcal{R}$ can be applied inside sequential composition of traces:

Proposition 4.12. *The following hold:*

- *If* $t_1 \xrightarrow{\mathsf{r}_i} t_1'$ *and* $t_1' \,;\, t_2$ *is defined, then* $t_1 \,;\, t_2 \xrightarrow{\mathsf{r}_i} t_1' \,;\, t_2$.
- *If* $t_2 \xrightarrow{\mathsf{r}_i} t_2'$ *and* $t_1 \,;\, t_2'$ *is defined, then* $t_1 \,;\, t_2 \xrightarrow{\mathsf{r}_i} t_1 \,;\, t_2'$.

From this property, we obtain the following proposition, which solves the case of sequential composition in the compositionality proof.

Proposition 4.13. *If* $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]$, *then* $\lfloor C_1 \,;\, C_2 \rfloor \subseteq [\![C_1' \,;\, C_2]\!]$. *Similarly, if* $\lfloor C_2 \rfloor \subseteq [\![C_2']\!]$, *then* $\lfloor C_1 \,;\, C_2 \rfloor \subseteq [\![C_1 \,;\, C_2']\!]$.

Proof. We prove the first claim and the second proof is symmetric. Suppose that $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]$. Let $t \in \lfloor C_1 \,;\, C_2 \rfloor$. By definition, we have $t = t_1 \,;\, t_2$ for some $t_1 \in \lfloor C_1 \rfloor$ and $t_2 \in \lfloor C_2 \rfloor$. Our assumption entails that $t_1 \in [\![C_1']\!]$. Let $t_1' \in \lfloor C_1' \rfloor$ such that $t_1' \xrightarrow{\mathcal{R}}{}^{*} t_1$. By Prop. 4.12, $t_1' \,;\, t_2 \xrightarrow{\mathcal{R}}{}^{*} t$. In particular, $t_1' \,;\, t_2$ is defined, and thus by definition we have $t_1' \,;\, t_2 \in \lfloor C_1' \,;\, C_2 \rfloor$. It follows that $t \in [\![C_1' \,;\, C_2]\!]$. □

Handling parallel composition is more difficult. Indeed, a claim like Prop. 4.12 does not hold for parallel composition instead of sequential composition: since the rewrite rules change the chronicle in the trace, it may be that $t_1 \xrightarrow{\mathsf{r}_i} t_1'$ and $t_1' \parallel t_2$ is defined, but $t_1 \parallel t_2$ is undefined. We address this problem by showing that in such cases there must be another trace t_2' such that $t_1 \parallel t_2' \xrightarrow{\mathcal{R}}{}^{*} t_1' \parallel t_2$ and, moreover, t_2' belongs to any concrete denotation that t_2 belongs to.

For the full argument, we introduce a set $\mathcal{D} \triangleq \{\xrightarrow{\mathsf{d}_1}, \xrightarrow{\mathsf{d}_2}, \xrightarrow{\mathsf{d}_3}, \xrightarrow{\mathsf{d}_4}\}$ of "dual" rewrite rules:

(1) $\langle s, \theta, c_1 \cdot \bar{\mathtt{W}}(x, v) \cdot c_2 \rangle \xrightarrow{\mathsf{d}_1} \langle s, \theta, c_1 \cdot e_1 \cdot \bar{\mathtt{W}}(x, v) \cdot e_2 \cdot c_2 \rangle$ provided that $e_1, e_2 \in \mathsf{EnvChro}$ and $(e_1 \cdot \bar{\mathtt{W}}(x, v) \cdot e_2)(c_1(s)) = c_1(s)[x \mapsto v]$.

(2) $\langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2 \rangle \xrightarrow{\mathsf{d}_2} \langle s, \theta, c_1 \cdot e_1 \cdot \mathtt{W}(x, v) \cdot e_2 \cdot c_2 \rangle$ provided that $e_1, e_2 \in \mathsf{EnvChro}$, $(e_1 \cdot \mathtt{W}(x, v) \cdot e_2)(c_1(s)) = c_1(s)[x \mapsto v]$, and $e_1(c_1(s))(x) = c_1(s)(x)$.

(3) $\langle s, \theta, c_1 \cdot c_2 \rangle \xrightarrow{\mathsf{d}_3} \langle s, \theta, c_1 \cdot \bar{\mathtt{W}}(x, v) \cdot c_2 \rangle$ provided that $c_1(s)(x) = v$.

(4) $\langle s, \theta, c_1 \cdot \bar{\mathtt{W}}(x, v) \cdot c_2 \rangle \xrightarrow{\mathsf{d}_4} \langle s, \theta, c_1 \cdot c_2 \rangle$ provided that $c_1(s)(x) = v$.

Proposition 4.14. *For every command* C, $\lfloor C \rfloor$ *is closed under* $\mathcal{D}$.

Proof. By induction on $t \in \lfloor C \rfloor$ using the following claims for the inductive step:

Claim 4.14.1: If $t_1 ; t_2 \xrightarrow{\mathrm{d}_i} t'$, then either $t' = t_1' ; t_2$ for some t_1' such that $t_1 \xrightarrow{\mathrm{d}_i} t_1'$ or $t' = t_1 ; t_2'$ for some t_2' such that $t_2 \xrightarrow{\mathrm{d}_i} t_2'$.

Claim 4.14.2: If $t_1 \parallel t_2 \xrightarrow{\mathrm{d}_i} t'$ for $i \in \{1, 3, 4\}$, then $t' = t_1' \parallel t_2'$ for some t_1' and t_2' such that $t_1 \xrightarrow{\mathrm{d}_i} t_1'$ and $t_2 \xrightarrow{\mathrm{d}_i} t_2'$.

Claim 4.14.3: If $t_1 \parallel t_2 \xrightarrow{\mathrm{d}_2} t'$, then $t' = t_1' \parallel t_2'$ for some t_1' and t_2' satisfying one of the following:

$$t_1 \xrightarrow{\mathrm{d}_2} t_1' \text{ and } t_2 \xrightarrow{\mathrm{d}_1} t_2' \quad \text{or} \quad t_1 \xrightarrow{\mathrm{d}_1} t_1' \text{ and } t_2 \xrightarrow{\mathrm{d}_2} t_2'.$$

□

Proposition 4.15. *Suppose that $t_1' \parallel t_2'$ is defined.*

- *If $t_1 \xrightarrow{\mathrm{r}_i} t_1'$, then there exists t_2 such that $t_2' \xrightarrow{\mathrm{d}_i} t_2$ and $t_1 \parallel t_2 \xrightarrow{\mathrm{r}_i} t_1' \parallel t_2'$.*
- *If $t_2 \xrightarrow{\mathrm{r}_i} t_2'$, then there exists t_1 such that $t_1' \xrightarrow{\mathrm{d}_i} t_1$ and $t_1 \parallel t_2 \xrightarrow{\mathrm{r}_i} t_1' \parallel t_2'$.*

With Propositions 4.14 and 4.15, we obtain the variant of Prop. 4.13 to handle parallel composition:

Proposition 4.16. *If $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]$, then $\lfloor C_1 \parallel C_2 \rfloor \subseteq [\![C_1' \parallel C_2]\!]$. Similarly, if $\lfloor C_2 \rfloor \subseteq [\![C_2']\!]$, then $\lfloor C_1 \parallel C_2 \rfloor \subseteq [\![C_1 \parallel C_2']\!]$.*

Proof. We prove the first claim and the second proof is symmetric. Suppose that $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]$. Let $t \in \lfloor C_1 \parallel C_2 \rfloor$. By definition, $t = t_1 \parallel t_2$ for some $t_1 \in \lfloor C_1 \rfloor$ and $t_2 \in \lfloor C_2 \rfloor$. Our assumption entails that $t_1 \in [\![C_1']\!]$. To show that $t \in [\![C_1' \parallel C_2]\!]$, it suffices to show that for every t_1' such that $t_1' \xrightarrow{\mathcal{R}}{}^* t_1$, there exists $t_2' \in \lfloor C_2 \rfloor$ such that $t_1' \parallel t_2' \xrightarrow{\mathcal{R}}{}^* t$. By Prop. 4.14, it suffices to show that for every t_1' such that $t_1' \xrightarrow{\mathcal{R}}{}^* t_1$, there exists t_2' such that $t_2 \xrightarrow{\mathcal{D}}{}^* t_2'$ and $t_1' \parallel t_2' \xrightarrow{\mathcal{R}}{}^* t$. We prove this claim by induction on the number of rewrite steps in $t_1' \xrightarrow{\mathcal{R}}{}^* t_1$. In the base case we have $t_1' = t_1$ and we can take $t_2' = t_2$ and $t_1' \parallel t_2' = t$. For the induction step, suppose that for t_1' there exists t_2' such that $t_2 \xrightarrow{\mathcal{D}}{}^* t_2'$ and $t_1' \parallel t_2' \xrightarrow{\mathcal{R}}{}^* t$, and let t_1'' such that $t_1'' \xrightarrow{\mathcal{R}} t_1'$. By Prop. 4.15, there exists t_2'' such that $t_2' \xrightarrow{\mathcal{D}} t_2''$ and $t_1'' \parallel t_2'' \xrightarrow{\mathcal{R}} t_1' \parallel t_2'$. Thus, we have $t_2 \xrightarrow{\mathcal{D}}{}^* t_2''$ and $t_1'' \parallel t_2'' \xrightarrow{\mathcal{R}}{}^* t$. □

Using Propositions 4.13 and 4.16 for handling sequential and parallel composition, and similar lemmas for other constructs, we can easily establish the following lemma by induction on P:

Lemma 4.17. *If $[\![C_1]\!] \subseteq [\![C_2]\!]$, then $[\![P[C_1]]\!] \subseteq [\![P[C_2]]\!]$ for every context P.*

As discussed above for the concrete semantics (see discussion after Lemma 3.3), the compositionality of $[\![\cdot]\!]$ follows from Lemma 4.17. This also entails that there exists a (mathematical) function that maps *the denotations* of the immediate sub-commands of C to the denotation of C. To see this, consider again the case of $C = C_1 \parallel C_2$. Given $[\![C_1]\!]$ and $[\![C_2]\!]$, we can arbitrarily "pick" some commands C_1' and C_2' with $[\![C_1']\!] = [\![C_1]\!]$ and $[\![C_2']\!] = [\![C_2]\!]$, and "return" $\lfloor C_1' \parallel C_2' \rfloor^{\mathcal{R}}$. Since $[\![C_1']\!] = [\![C_1]\!]$ and $[\![C_2']\!] = [\![C_2]\!]$, the compositionality of $[\![\cdot]\!]$ ensures that $[\![C]\!] = [\![C_1 \parallel C_2]\!] = [\![C_1' \parallel C_2']\!] = \lfloor C_1' \parallel C_2' \rfloor^{\mathcal{R}}$.

Remark 4.18. Candidates for a direct compositional definition of $[\![C_1 ; C_2]\!]$ and $[\![C_1 \parallel C_2]\!]$ are to take the $\mathcal{R}$-closure of the set obtained by taking all possible sequential/parallel compositions of traces from $[\![C_1]\!]$ and $[\![C_2]\!]$. This works for sequential composition, as we have $[\![C_1 ; C_2]\!] = \{t_1 ; t_2 \mid t_1 \in [\![C_1]\!], t_2 \in [\![C_2]\!]\}^{\mathcal{R}}$. However, for parallel composition, we only have $[\![C_1 \parallel C_2]\!] \subseteq \{t_1 \parallel t_2 \mid t_1 \in [\![C_1]\!], t_2 \in [\![C_2]\!]\}^{\mathcal{R}}$. To see that the converse does not hold, let:

$$C_1 = \mathtt{x := 1 ; assume(y = 0) ; assume(z \neq 1) ; assume(z = 1) ; x := 0}$$

$$C_2 = \mathtt{y := 1 ; assume(x = 0) ; assume(z \neq 1) ; assume(z = 1) ; y := 0}$$

Store-before-store elimination

$x := E\,;x := E' \rightsquigarrow x := E'$

Store-after-load eliminations

$\texttt{let}\ a = x\ \texttt{in}\ (x := a\,;C) \rightsquigarrow \texttt{let}\ a = x\ \texttt{in}\ C$

$\texttt{let}\ a = x\ \texttt{in}\ (\texttt{if}\ E\ \texttt{then}\ (x := a\,;C_1)\ \texttt{else}\ C_2) \rightsquigarrow \texttt{let}\ a = x\ \texttt{in}\ (\texttt{if}\ E\ \texttt{then}\ C_1\ \texttt{else}\ C_2)$

$\texttt{let}\ a = x\ \texttt{in}\ (y := E\,;x := a) \rightsquigarrow y := E$ provided that $x \neq y$ and $a \notin \mathsf{fv}(E)$

Unused FAA-before-store elimination

$\texttt{let}\ a = \mathsf{FAA}(x, E)\ \texttt{in}\ x := v \rightsquigarrow x := v$

FAA-after-FAA elimination

$\texttt{let}\ a = \mathsf{FAA}(x, E)\ \texttt{in}\ \texttt{let}\ b = \mathsf{FAA}(x, E')\ \texttt{in}\ C \rightsquigarrow \texttt{let}\ a = \mathsf{FAA}(x, E + E')\ \texttt{in}\ \texttt{let}\ b = a + E\ \texttt{in}\ C$ provided that $a \notin \mathsf{fv}(E) \cup \mathsf{fv}(E')$

FAA-after-store elimination

$x := v\,;(\texttt{let}\ a = \mathsf{FAA}(x, E)\ \texttt{in}\ C) \rightsquigarrow \texttt{let}\ a = v\ \texttt{in}\ (x := v + [\![E]\!]\,;C)$ provided that $a \notin \mathsf{fv}(E)$

Redundant FAA elimination/introduction

$\texttt{let}\ a = x\ \texttt{in}\ C \leftrightsquigarrow \texttt{let}\ a = \mathsf{FAA}(x, 0)\ \texttt{in}\ C$

Fig. 5. Examples of program transformations validated by the abstract semantics

Using r_2 on $\langle \mathsf{s}_0, \theta_0, \mathsf{W}(\mathsf{x}, 1) \cdot \bar{\mathsf{W}}(\mathsf{z}, 1) \cdot \mathsf{W}(\mathsf{x}, 0)\rangle \in \lfloor C_1 \rfloor$ and $\langle \mathsf{s}_0, \theta_0, \mathsf{W}(\mathsf{y}, 1) \cdot \bar{\mathsf{W}}(\mathsf{z}, 1) \cdot \mathsf{W}(\mathsf{y}, 0)\rangle \in \lfloor C_2 \rfloor$, we obtain that $\langle \mathsf{s}_0, \theta_0, \bar{\mathsf{W}}(\mathsf{z}, 1)\rangle \in [\![C_1]\!] \parallel [\![C_2]\!]$. But, $\langle \mathsf{s}_0, \theta_0, \bar{\mathsf{W}}(\mathsf{z}, 1)\rangle \notin \{t_1 \parallel t_2 \mid t_1 \in [\![C_1]\!], t_2 \in [\![C_2]\!]\}^{\mathcal{R}}$. Indeed, no trace in $\lfloor C_1 \parallel C_2 \rfloor$ has a single environment write to z, and all rules of $\mathcal{R}$ preserve the environment actions.

4.2 Adequacy

We show that adequacy of the abstract semantics is a corollary of its compositionality and of Lemma 3.4. For that, we first observe that the rewrite rules only manipulate chronicles leaving the initial state, initial store, and (derived) final state intact, and that only component traces are mapped to component traces by the rewrite rules in $\mathcal{R}$.

Proposition 4.19. *Suppose that* $\langle s, \theta, c\rangle \xrightarrow{\mathcal{R}} \langle s', \theta', c'\rangle$. *Then,* $s = s'$, $\theta = \theta'$, *and* $c(s) = c'(s')$.

Proposition 4.20. *If* $\langle s, \theta, c\rangle \xrightarrow{\mathcal{R}} \langle s', \theta', c'\rangle$ *and* $c' \in \mathsf{CmpChro}$, *then* $c \in \mathsf{CmpChro}$.

Theorem 4.21. *If* $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$, *then* $C_{\mathrm{src}} \rightsquigarrow C_{\mathrm{tgt}}$.

Proof. Suppose that $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$. Let P be a context such that $P[C_{\mathrm{src}}]$ and $P[C_{\mathrm{tgt}}]$ are closed, and suppose that $\langle P[C_{\mathrm{tgt}}], s\rangle \downarrow s'$. Since $\langle P[C_{\mathrm{tgt}}], s\rangle \downarrow s'$, by Lemma 3.4, we have $\langle s, \theta, c\rangle \in \lfloor P[C_{\mathrm{tgt}}] \rfloor$ for some store θ and component chronicle $c \in \mathsf{CmpChro}$ such that $c(s) = s'$. Since $\lfloor P[C_{\mathrm{tgt}}] \rfloor \subseteq [\![P[C_{\mathrm{tgt}}]]\!]$, we have $\langle s, \theta, c\rangle \in [\![P[C_{\mathrm{tgt}}]]\!]$. Since $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$, by Lemma 4.17, it follows that $\langle s, \theta, c\rangle \in [\![P[C_{\mathrm{src}}]]\!]$. Using Prop. 4.2, $t_0 \xrightarrow{\mathcal{R}}{}^{*} \langle s, \theta, c\rangle$ for some $t_0 \in \lfloor P[C_{\mathrm{src}}] \rfloor$. Then, by Propositions 4.19 and 4.20, $t_0 = \langle s, \theta, c'\rangle$ for some component chronicle $c' \in \mathsf{CmpChro}$ such that $c'(s) = c(s) = s'$. By Lemma 3.4, it follows that $\langle P[C_{\mathrm{src}}], s\rangle \downarrow s'$. □

By Prop. 4.3, we have $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$ iff $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq [\![C_{\mathrm{src}}]\!]$. It follows that every program transformation that is validated by the concrete semantics is also validated by the abstract one:

Proposition 4.22. *If* $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq \lfloor C_{\mathrm{src}} \rfloor$, *then* $[\![C_{\mathrm{tgt}}]\!] \subseteq [\![C_{\mathrm{src}}]\!]$.

Figure 5 presents examples of refinements that are validated by the abstract semantics but not by the concrete semantics. Again, among RMWs, we only list transformations involving FAA.

4.3 Full Abstraction

We establish full abstraction for the abstract semantics. The proof uses the notation $\bar{c}$ for the *dual* of a chronicle c, defined by $\bar{c} \triangleq \bar{\alpha_1} \cdots \bar{\alpha_n}$ for $c = \alpha_1 \cdots \alpha_n$.

THEOREM 4.23. *If* $[\![C_{\text{tgt}}]\!] \not\subseteq [\![C_{\text{src}}]\!]$, *then* $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$.

PROOF. Suppose that $[\![C_{\text{tgt}}]\!] \not\subseteq [\![C_{\text{src}}]\!]$. By Prop. 4.3, we have $\lfloor C_{\text{tgt}} \rfloor \not\subseteq [\![C_{\text{src}}]\!]$. Let $t_{\text{tgt}} = \langle s_0, \theta, c_{\text{tgt}} \rangle \in \lfloor C_{\text{tgt}} \rfloor \setminus [\![C_{\text{src}}]\!]$. Using t_{tgt}, we construct a context that demonstrates $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$. Suppose first that t_{tgt} is non-empty, and let $\alpha_1, \ldots, \alpha_n \in \text{Act}$ such that $c_{\text{tgt}} = \alpha_1 \cdots \alpha_n$. For every $1 \leq i \leq n$, let:

$$s_i \triangleq (\alpha_1 \cdots \alpha_i)(s_0) \qquad C_i \triangleq \begin{cases} \texttt{let a = XCHG}(x, v)\ \texttt{in assume(a =}\ s_{i-1}(x)\texttt{)} & \alpha_i = \bar{\mathtt{W}}(x, v) \\ \texttt{skip} & \text{otherwise} \end{cases}$$

Let $a_1, \ldots, a_k$ be an enumeration of $\text{fv}(C_{\text{src}}) \cup \text{fv}(C_{\text{tgt}})$, and define:

$$C_{\text{ctx}} \triangleq C_1 \,;\, \texttt{snapshot}(s_1) \,;\, C_2 \,;\, \texttt{snapshot}(s_2) \,;\, \ldots \,;\, C_{n-1} \,;\, \texttt{snapshot}(s_{n-1}) \,;\, C_n$$
$$P \triangleq \texttt{let}\ a_1 = \theta(a_1)\ \texttt{in}\ (\texttt{let}\ a_2 = \theta(a_2)\ \texttt{in}\ (\ldots (\texttt{let}\ a_k = \theta(a_k)\ \texttt{in}\ (C_{\text{ctx}} \parallel -)))$$

Clearly, $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed. We claim that (i) $\langle P[C_{\text{tgt}}], s_0 \rangle \downarrow s_n$, but (ii) $\langle P[C_{\text{src}}], s_0 \rangle \not\downarrow s_n$. For (i), observe that $\langle s_0, \theta, c_{\text{tgt}}^{-} \parallel c_{\text{tgt}} \rangle \in \lfloor P[C_{\text{tgt}}] \rfloor$ for any store θ. Then, since $c_{\text{tgt}}^{-} \parallel c_{\text{tgt}}$ is a component chronicle and $c_{\text{tgt}}(s_0) = s_n$, $\langle P[C_{\text{tgt}}], s_0 \rangle \downarrow s_n$ follows by Lemma 3.4.

To prove (ii), it suffices to prove the following claim:

Claim 4.23.1: $\langle s_0, \theta, \bar{c} \rangle \xrightarrow{\mathcal{R}}{}^{*} \langle s_0, \theta, c_{\text{tgt}} \rangle$ for every $c \in \text{Chro}$ such that $c(s_0) = s_n$ and $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$.

Indeed, from this claim we obtain that $\langle s_0, \theta, \bar{c} \rangle \notin \lfloor C_{\text{src}} \rfloor$ for every chronicle c such that $c(s_0) = s_n$ and $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$, which implies that $\langle s_0, \theta, c \rangle \notin \lfloor P[C_{\text{src}}] \rfloor$ for every component chronicle c satisfying $c(s_0) = s_n$. Then, (ii) follows by Lemma 3.4.

Next, we prove Claim 4.23.1. Let c be a chronicle such that $c(s_0) = s_n$ and $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$. Due to the use of snapshots in C_{ctx}, since $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$, we have that $c = c_1 \cdots c_n$ for some chronicles $c_1, \ldots, c_n$ such that $c_i(s_{i-1}) = s_i$ and $\langle s_{i-1}, \theta, c_i \rangle \in \lfloor C_i \rfloor$ for every $1 \leq i \leq n$. We show that for every $1 \leq i \leq n$, we have $\langle s_{i-1}, \theta, \bar{c_i} \rangle \xrightarrow{\mathcal{R}}{}^{*} \langle s_{i-1}, \theta, \alpha_i \rangle$. By repeatedly applying this rewrite, using Prop. 4.12, the desired $\langle s_0, \theta, \bar{c} \rangle \xrightarrow{\mathcal{R}}{}^{*} \langle s_0, \theta, c_{\text{tgt}} \rangle$ follows.

Let $1 \leq i \leq n$, and consider the possible cases:

- $\alpha_i = \bar{\mathtt{W}}(x, v)$ is an environment write: In this case, $\langle s_{i-1}, \theta, c_i \rangle \in \lfloor C_i \rfloor$ implies that there exists environment chronicles e', e such that $c_i = e' \cdot \mathtt{W}(x, v) \cdot e$ and $e'(s_{i-1})(x) = s_{i-1}(x)$. Then, since we also have $c_i(s_{i-1}) = s_i$, using r_2, we can rewrite $\langle s_{i-1}, \theta, \bar{c_i} \rangle$ into $\langle s_{i-1}, \theta, \alpha_i \rangle$.
- $\alpha_i = \mathtt{W}(x, v)$ is an component write: In this case, either $c_i = e' \cdot \bar{\mathtt{W}}(x, v) \cdot e$ for some environment chronicles e', e or $\bar{\alpha_i}$ is not inside c_i, and c_i is an environment chronicle. In the first case, since $c_i(s_{i-1}) = s_i$, using r_1, we can rewrite $\langle s_{i-1}, \theta, \bar{c_i} \rangle$ into $\langle s_{i-1}, \theta, \alpha_i \rangle$. In the second case, $c_i(s_{i-1}) = s_i = \alpha_i(s_{i-1})$ implies that $s_{i-1} = s_i$. Using r_4, we can rewrite $\langle s_{i-1}, \theta, \bar{c_i} \rangle$ into $\langle s_{i-1}, \theta, \alpha_i \cdot \bar{c_i} \rangle$. Then, using the r_3' rewrite rule (a combination of r_1 and r_4 and r_3, see Remark 4.11), we rewrite $\langle s_{i-1}, \theta, \alpha_i \cdot \bar{c_i} \rangle$ into $\langle s_{i-1}, \theta, \alpha_i \rangle$.

Finally, consider the case that $c_{\text{tgt}} = \varepsilon$. In this case we define $C_{\text{ctx}} \triangleq \texttt{skip}$ and also define P as above (using C_{ctx}). We have $\langle s_0, \theta, \varepsilon \rangle \in \lfloor P[C_{\text{tgt}}] \rfloor$ for any store θ, and by Lemma 3.4 we obtain that $\langle P[C_{\text{tgt}}], s_0 \rangle \downarrow s_0$. In turn, as above, $\langle P[C_{\text{src}}], s_0 \rangle \not\downarrow s_0$ follows from the fact that $\langle s_0, \theta, \bar{c} \rangle \xrightarrow{\mathcal{R}}{}^{*} \langle s_0, \theta, \varepsilon \rangle$ for every chronicle c such that $c(s_0) = s_0$ and $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$. To prove this fact, let c such that $c(s_0) = s_0$ and $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$. Then, $\langle s_0, \theta, c \rangle \in \lfloor C_{\text{ctx}} \rfloor$ implies that c is an environment chronicle. Using r_3', we rewrite $\langle s_0, \theta, \bar{c} \rangle$ into $\langle s_0, \theta, \varepsilon \rangle$. □

Given Thm. 4.23, we can use the abstract semantics to easily *invalidate* certain transformations. Next, we present two such examples.

Example 4.24. Store-before-store elimination is not valid when a (used) load appears between two stores. For instance, for $C_{\text{src}} = \texttt{x := 1 ; let a = y in (x := 2 ; z := a)}$ and $C_{\text{tgt}} = \texttt{let a = y in (x := 2 ; z := a)}$, we have $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$. Observing that $t = \langle s_0, \theta_0, \bar{\mathtt{W}}(\mathtt{y}, 1) \cdot \mathtt{W}(\mathtt{x}, 2) \cdot \mathtt{W}(\mathtt{z}, 0)\rangle$ satisfies $t \in [\![C_{\text{tgt}}]\!] \setminus [\![C_{\text{src}}]\!]$, this follows from Thm. 4.23.

Example 4.25. A repeated store cannot be eliminated when there is another store between the repeated stores. For instance, for $C_{\text{src}} = \texttt{x := 1 ; y := 1 ; x := 1}$ and $C_{\text{tgt}} = \texttt{x := 1 ; y := 1}$, we have $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$. Observing that $t = \langle s_0, \theta_0, \mathtt{W}(\mathtt{x}, 1) \cdot \bar{\mathtt{W}}(\mathtt{x}, 2) \cdot \mathtt{W}(\mathtt{y}, 1)\rangle$ satisfies $t \in [\![C_{\text{tgt}}]\!] \setminus [\![C_{\text{src}}]\!]$, this follows from Thm. 4.23.

From the full abstraction proof, we observe that although multiple rewrites of a trace may be necessary, these rewrites do not overlap. We only apply them to disjoint parts of the chronicle. Formally, we let $\mathcal{R}^{\text{alt}} \triangleq \{\xrightarrow{\mathtt{r}_1^{\text{alt}}}, \xrightarrow{\mathtt{r}_2^{\text{alt}}}, \xrightarrow{\mathtt{r}_3^{\text{alt}}}, \xrightarrow{\mathtt{r}_4^{\text{alt}}}\}$ be the set consisting of "local" variants of the rules:

(1) $\langle s, \theta, m_1 \cdot \mathtt{W}(x, v) \cdot m_2\rangle \xrightarrow{\mathtt{r}_1^{\text{alt}}} \langle s, \theta, \mathtt{W}(x, v)\rangle$ provided that $m_1, m_2 \in \mathsf{CmpChro}$ and $(m_1 \cdot \mathtt{W}(x, v) \cdot m_2)(s) = s[x \mapsto v]$.

(2) $\langle s, \theta, m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2\rangle \xrightarrow{\mathtt{r}_2^{\text{alt}}} \langle s, \theta, \bar{\mathtt{W}}(x, v)\rangle$ provided that $m_1, m_2 \in \mathsf{CmpChro}$, $(m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2)(s) = s[x \mapsto v]$, and $m_1(s)(x) = s(x)$.

(3) $\langle s, \theta, m\rangle \xrightarrow{\mathtt{r}_3^{\text{alt}}} \langle s, \theta, \varepsilon\rangle$ provided that $m \in \mathsf{CmpChro}$ and $m(s) = s$.

(4) $\langle s, \theta, \varepsilon\rangle \xrightarrow{\mathtt{r}_4^{\text{alt}}} \langle s, \theta, \mathtt{W}(x, v)\rangle$ provided that $s(x) = v$.

A relation $\rightrightarrows$ between traces is inductively defined as follows:

$$\frac{}{\langle s, \theta, \varepsilon\rangle \rightrightarrows \langle s, \theta, \varepsilon\rangle} \qquad \frac{t_1 \xrightarrow{\mathcal{R}^{\text{alt}}{}^{?}} t_1' \qquad t_2 \rightrightarrows t_2'}{t_1 \,;\, t_2 \rightrightarrows t_1' \,;\, t_2'}$$

Then, denoting $T^{\rightrightarrows} \triangleq \{t' \mid \exists t \in T.\, t \rightrightarrows t'\}$, the full abstraction proof actually shows that $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$ whenever $\lfloor C_{\text{tgt}} \rfloor \not\subseteq \lfloor C_{\text{src}} \rfloor^{\rightrightarrows}$. In fact, by analyzing the rewrite rules we can prove the following (the proof is given in §A.2):

Lemma 4.26. *For every set T of traces, $T^{\rightrightarrows} = T^{\mathcal{R}}$.*

4.4 Full Abstraction Without Snapshots

The full abstraction proof above relies on the availability of the `snapshot(s)` command, which gives the parallel context the ability to simultaneously observe the values of all variables. Next, we show that snapshots can be avoided in that proof given that C_{src} is loop-free. This means that when C_{src} is loop-free snapshots do no not increase the distinguishing power of the parallel context. In turn, we present a delicate example of a command C_{src} with loops, where a certain refinement holds for snapshot-free contexts but fail to hold by when the context has snapshots. In particular, this implies that full abstraction of the abstract semantics does not hold for code fragments with loops in a language without snapshot.

Formally, we say that a command C is *loop-free* if C contains no while commands, and that a context P is *snapshot-free* if P contains no snapshot commands. The transformation from a command C_{src} to a command C_{tgt} is *sound for no-snapshot context*, denoted by $C_{\text{src}} \rightsquigarrow_{\not\text{snapshot}} C_{\text{tgt}}$, if for every snapshot-free context P such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, we have that $\langle P[C_{\text{tgt}}], s\rangle \downarrow s'$ implies $\langle P[C_{\text{src}}], s\rangle \downarrow s'$.

Theorem 4.27. *If C_{src} is loop-free and $[\![C_{\mathrm{tgt}}]\!] \not\subseteq [\![C_{\mathrm{src}}]\!]$, then $C_{\mathrm{src}} \not\rightsquigarrow_{\not\text{snapshot}} C_{\mathrm{tgt}}$.*

Proof Sketch. In the proof Thm. 4.23, snapshot is needed in order to ensure that a certain state is reached when C_{src} is executed concurrently. When C_{src} is loop-free, we can achieve this result by repeatedly reading the shared variables used in C_{src}, and checking their values one-by-one. More precisely, given a state s, let $C_s \triangleq \mathtt{assume}(x_1 = s(x_1)) ; \ldots ; \mathtt{assume}(x_n = s(x_n))$ where $x_1, \ldots, x_n$ is an enumeration of all shared variables occurring in C_{src}. When C_{src} is loop-free, there exists a bound $N \in \mathbb{N}$ on the number of writes performed by C_{src} (i.e., the number of component actions in $\lfloor C_{\mathrm{src}} \rfloor$). We use a sequential composition $C_s ; \ldots ; C_s$ consisting of $N+1$ copies of C_s instead of $\mathtt{snapshot}(s)$. If after every execution of C_s we reach a state different than s, then for the next execution of C_s to terminate, we need at least one write by the concurrent context. Since the C_{src} is performing at most N writes, executing C_s $N+1$ times in a row ensures that at some point we visit s. □

The above implication fails if C_{src} has loops. The simplest example we found is presented next.

Example 4.28. For the commands $C_{\mathrm{src}} = \mathtt{while} * \mathtt{do} (\mathtt{y} := 0 ; \mathtt{x} := * ; \mathtt{x} := 0 ; \mathtt{y} := *)$ and $C_{\mathrm{tgt}} = \mathtt{y} := 0 ; \mathtt{x} := 1 ; \mathtt{y} := 1 ; \mathtt{x} := 0$, we have $C_{\mathrm{src}} \not\rightsquigarrow C_{\mathrm{tgt}}$ but $C_{\mathrm{src}} \rightsquigarrow_{\not\text{snapshot}} C_{\mathrm{tgt}}$. The former follows from Thm. 4.23 since we have $\langle \mathsf{s}_0, \theta_0, \mathtt{W}(\mathtt{y}, 0) \cdot \mathtt{W}(\mathtt{x}, 1) \cdot \mathtt{W}(\mathtt{y}, 1) \cdot \mathtt{W}(\mathtt{x}, 0) \rangle \in \lfloor C_{\mathrm{tgt}} \rfloor \setminus [\![C_{\mathrm{src}}]\!]$.

To see that $C_{\mathrm{src}} \rightsquigarrow_{\not\text{snapshot}} C_{\mathrm{tgt}}$, we have to resort to cumbersome operational reasoning, and provide a simulation relation that relates operational executions of $P[C_{\mathrm{tgt}}]$ to those of $P[C_{\mathrm{src}}]$. Roughly speaking, the main idea is to execute $\mathtt{y} := 0$ and $\mathtt{x} := *$ (with 1 for $*$) in the source when the target executes $\mathtt{y} := 0$ and $\mathtt{x} := 1$, respectively. Then, when the target executes $\mathtt{y} := 1$, the source executes $\mathtt{x} := 0 ; \mathtt{y} := *$ (with 1 for $*$). This creates a mismatch between the target's state that has $\mathtt{x} = 1$ and the source's state that has $\mathtt{x} = 0$. Nevertheless, whenever the concurrent context relies on the value of $\mathtt{x}$, the source can do another half-iteration and execute $\mathtt{y} := 0 ; \mathtt{x} := *$ to fix the value of $\mathtt{x}$ as it is in the target's state, moving the mismatch between the target and the source to $\mathtt{y}$. This way, we are able to use the source's non-deterministic loop, to provide the concurrent context with whatever value it needs for $\mathtt{x}$ and $\mathtt{y}$, one at a time. Finally, when the target executes $\mathtt{x} := 0$ the source executes $\mathtt{x} := 0 ; \mathtt{y} := *$ (with the final value of $\mathtt{y}$ in the target for $*$).

The process of making this intuition formal is rather complex, and requires to generalize the notion of a command context, and understand how generalized contexts interact with the operational semantics. We refer the reader to §A.1 in the supplementary material. The fact that this process is so challenging provides us with more confidence that the denotational semantics can be very beneficial in the formal study of contextual refinements.

Example 4.28 uses non-deterministic looping, $\mathtt{while} * \mathtt{do}\ C$. To get rid of non-deterministic looping, one can use the following transformations:

Proposition 4.29. *The following transformations are sound:*

- $x := * ; \mathtt{while}\ x\ \mathtt{do}\ (x := * ; C ; x := *) ; x := * \rightsquigarrow \mathtt{while} * \mathtt{do}\ C$.
- *For* $C = \mathtt{while}\ y\ \mathtt{do}\ (x := 0 \oplus \mathtt{let}\ a = \mathtt{FAA}(x, 1)\ \mathtt{in}\ \mathtt{skip})$, *we have* $\mathtt{let}\ a = y\ \mathtt{in}\ (y := 1 ; (C \parallel y := 0) ; y := a) \rightsquigarrow x := *$ *provided that* $x \neq y$.

Then, using Prop. 4.29, we can adapt Example 4.28 to use a command C'_{src} that does not use $\mathtt{while} * \mathtt{do}\ C$ and $x := *$ instead of C_{src} and have $C'_{\mathrm{src}} \rightsquigarrow_{\not\text{snapshot}} C_{\mathrm{tgt}}$. In turn, to see that $C'_{\mathrm{src}} \not\rightsquigarrow C_{\mathrm{tgt}}$, note that a concurrent snapshot observing $\mathtt{x} = \mathtt{y} = 1$ is only possible for C_{tgt} but not for C'_{src}. It follows that snapshots strictly increase the distinguish power of parallel contexts also in a language without $\mathtt{while} * \mathtt{do}\ C$ and $x := *$.

5 SEMANTICS FOR RMW-FREE CONTEXTS

In this section we show that RMWs strictly increase the power of the context to distinguish between code fragments, and identify how the abstract semantics can be adapted under the assumption that the context does not perform RMW instructions.

Formally, we say that a command C (respectively, context P) is *RMW-free* if C (respectively, P) contains no RMW commands. A transformation from a command C_{src} to a command C_{tgt} is *sound for no-RMW context*, denoted by $C_{\text{src}} \rightsquigarrow_{\not\text{rmw}} C_{\text{tgt}}$, if for every RMW-free context P such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, we have that $\langle P[C_{\text{tgt}}], s\rangle \downarrow s'$ implies $\langle P[C_{\text{src}}], s\rangle \downarrow s'$.

The next example demonstrates a case where $C_{\text{src}} \rightsquigarrow_{\not\text{rmw}} C_{\text{tgt}}$ but $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$. (Example 4.7 provides another case in point.)

Example 5.1. Let:

$$C_{\text{src}} = \mathtt{x} := 1 \,;\, \mathtt{x} := 5 \,;\, \mathtt{if\ x = 2\ then\ x := 3\ else\ x := 4}$$
$$C_{\text{tgt}} = \mathtt{x} := 1 \,;\, \mathtt{if\ x = 2\ then\ x := 3\ else\ x := 4}$$

An intuitive argument for $C_{\text{src}} \rightsquigarrow C_{\text{tgt}}$ could claim that when the "then" branch is taken, there must be a moment when the parallel context stores 2 in x and we can execute x := 5 "just before" that moment; and when the "else" branch is taken we can execute x := 5;if x = 2 then x := 3 else x := 4 as one atomic block at the time the target executes x := 4. This argument, however, ignores the option that the context may not be able to store 2 in x if the value of x was modified to 5, which is possible when the context stores 2 in x using an RMW. Indeed, for $P \triangleq - \parallel \mathtt{let\ b = FAA(x, 1)\ in\ skip}$ we have $\langle P[C], \mathtt{s}_0\rangle \downarrow \mathtt{s}_0[\mathtt{x} \mapsto 3]$ for $C = C_{\text{tgt}}$ but not for $C = C_{\text{src}}$. Alternatively, using Thm. 4.23, $C_{\text{src}} \not\rightsquigarrow C_{\text{tgt}}$ follows from the fact that for $t \triangleq \langle \mathtt{s}_0, \theta_0, \mathtt{W(x,1)} \cdot \bar{\mathtt{W}}\mathtt{(x,2)} \cdot \mathtt{W(x,3)}\rangle$, we have $t \in [\![C_{\text{tgt}}]\!] \setminus [\![C_{\text{src}}]\!]$. Using the semantics below, we will formally show that $C_{\text{src}} \rightsquigarrow_{\not\text{rmw}} C_{\text{tgt}}$.

From the discussion above, we observe that when the context is RMW-free, we would like to allow to attach component actions to environment actions even when the component actions write to the same variable that the environment modifies. This would formally justify the intuitive argument about the "then" branch, and will allows us to rewrite $\langle \mathtt{s}_0, \theta_0, \mathtt{W(x,1)} \cdot \bar{\mathtt{W}}\mathtt{(x,2)} \cdot \mathtt{W(x,3)}\rangle$ into the target trace $\langle \mathtt{s}_0, \theta_0, \mathtt{W(x,1)} \cdot \mathtt{W(x,3)}\rangle$. We do so by omitting the side condition of $\mathtt{r}_2$, using the following strengthening (in the sense that it allows more rewrites) of $\mathtt{r}_2$ instead:

(2) $\langle s, \theta, c_1 \cdot m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2 \cdot c_2\rangle \xrightarrow{\mathtt{r}_2^{\not\text{rmw}}} \langle s, \theta, c_1 \cdot \bar{\mathtt{W}}(x, v) \cdot c_2\rangle$ provided that $m_1, m_2 \in \mathsf{CmpChro}$ and $(m_1 \cdot \bar{\mathtt{W}}(x, v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$.

We define $\mathcal{R}_{\not\text{rmw}} \triangleq \{\xrightarrow{\mathtt{r}_1}, \xrightarrow{\mathtt{r}_2^{\not\text{rmw}}}, \xrightarrow{\mathtt{r}_3}, \xrightarrow{\mathtt{r}_4}\}$ and $[\![C]\!]_{\not\text{rmw}} \triangleq \lfloor C \rfloor^{\mathcal{R}_{\not\text{rmw}}}$.

Next, we prove the following compositionality property, analogous to 4.17:

Lemma 5.2. *If* $[\![C_1]\!]_{\not\text{rmw}} \subseteq [\![C_2]\!]_{\not\text{rmw}}$, *then* $[\![P[C_1]]\!] \subseteq [\![P[C_2]]\!]$ *for every RMW-free context* P.

Proof. The proof proceeds by induction on P. For sequential composition, we use the following analogue of Prop. 4.13.

Claim 5.2.1: If $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]_{\not\text{rmw}}$, then $\lfloor C_1 \,;\, C_2 \rfloor \subseteq [\![C_1' \,;\, C_2]\!]_{\not\text{rmw}}$. Similarly, if $\lfloor C_2 \rfloor \subseteq [\![C_2']\!]_{\not\text{rmw}}$, then $\lfloor C_1 \,;\, C_2 \rfloor \subseteq [\![C_1 \,;\, C_2']\!]_{\not\text{rmw}}$.

For parallel composition, we define the following rule that is dual to $\mathtt{r}_2^{\not\text{rmw}}$:

- $\langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2\rangle \xrightarrow{\mathtt{d}_2^{\not\text{rmw}}} \langle s, \theta, c_1 \cdot e_1 \cdot \mathtt{W}(x, v) \cdot e_2 \cdot c_2\rangle$ provided that $e_1, e_2 \in \mathsf{EnvChro}$ and $(e_1 \cdot \mathtt{W}(x, v) \cdot e_2)(c_1(s)) = c_1(s)[x \mapsto v]$.

We define $\mathcal{D}_{\not\text{rmw}} \triangleq \{\xrightarrow{\mathtt{d}_1}, \xrightarrow{\mathtt{d}_2^{\not\text{rmw}}}, \xrightarrow{\mathtt{d}_3}, \xrightarrow{\mathtt{d}_4}\}$, and show the following analogue of Prop. 4.14:

Claim 5.2.2: For every RMW-free command C, $\lfloor C \rfloor$ is closed under $\mathcal{D}^{\not\text{rmw}}$.

Then, we can prove the following variant of Prop. 4.16, which establishes the required property for parallel composition:

Claim 5.2.3: If $\lfloor C_1 \rfloor \subseteq [\![C_1']\!]_{\not{\mathsf{rmw}}}$ and C_2 is RMW-free, then $\lfloor C_1 \parallel C_2 \rfloor \subseteq [\![C_1' \parallel C_2]\!]_{\not{\mathsf{rmw}}}$. Similarly, if $\lfloor C_2 \rfloor \subseteq [\![C_2']\!]_{\not{\mathsf{rmw}}}$ and C_1 is RMW-free, then $\lfloor C_1 \parallel C_2 \rfloor \subseteq [\![C_1 \parallel C_2']\!]_{\not{\mathsf{rmw}}}$.

Other language constructs are handled similarly. □

Given Lemma 5.2, adequacy of $[\![\cdot]\!]_{\not{\mathsf{rmw}}}$ is shown similarly to the proof of Thm. 4.21.

Theorem 5.3. *If* $[\![C_{\mathrm{tgt}}]\!]_{\not{\mathsf{rmw}}} \subseteq [\![C_{\mathrm{src}}]\!]_{\not{\mathsf{rmw}}}$, *then* $C_{\mathrm{src}} \rightsquigarrow_{\not{\mathsf{rmw}}} C_{\mathrm{tgt}}$.

With Thm. 5.3, we can revisit Example 5.1 and derive $C_{\mathrm{src}} \rightsquigarrow_{\not{\mathsf{rmw}}} C_{\mathrm{tgt}}$ from $\lfloor C_{\mathrm{tgt}} \rfloor \subseteq [\![C_{\mathrm{src}}]\!]_{\not{\mathsf{rmw}}}$. Indeed, traces in $\lfloor C_{\mathrm{tgt}} \rfloor$ are either of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \cdot \bar{\mathtt{W}}(\mathsf{x}, 2) \cdot e_3 \cdot \mathtt{W}(\mathsf{x}, 3) \cdot e_4 \rangle$ or of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \cdot \mathtt{W}(\mathsf{x}, 4) \cdot e_3 \rangle$. For t of the first form, we can start from a corresponding trace in $\lfloor C_{\mathrm{src}} \rfloor$ of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \cdot \mathtt{W}(\mathsf{x}, 5) \cdot \bar{\mathtt{W}}(\mathsf{x}, 2) \cdot e_3 \cdot \mathtt{W}(\mathsf{x}, 3) \cdot e_4 \rangle$ and rewrite by $\mathsf{r}_2^{\not{\mathsf{rmw}}}$ with $c_1 = e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2$, $m_1 = \mathtt{W}(\mathsf{x}, 5)$, $m_2 = \varepsilon$, and $c_2 = e_3 \cdot \mathtt{W}(\mathsf{x}, 3) \cdot e_4$ to obtain t. For t of the second form, we can start from a corresponding trace in $\lfloor C_{\mathrm{src}} \rfloor$ of the form $\langle s, \theta, e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2 \cdot \mathtt{W}(\mathsf{x}, 5) \cdot \mathtt{W}(\mathsf{x}, 4) \cdot e_3 \rangle$ and rewrite by r_1 with $c_1 = e_1 \cdot \mathtt{W}(\mathsf{x}, 1) \cdot e_2$, $m_1 = \mathtt{W}(\mathsf{x}, 5)$, $m_2 = \varepsilon$, and $c_2 = e_3$ to obtain t.

Finally, we establish a full abstraction properties for the ~~rmw~~ semantics.

Theorem 5.4. *If* $[\![C_{\mathrm{tgt}}]\!]_{\not{\mathsf{rmw}}} \not\subseteq [\![C_{\mathrm{src}}]\!]_{\not{\mathsf{rmw}}}$, *then* $C_{\mathrm{src}} \not\rightsquigarrow_{\not{\mathsf{rmw}}} C_{\mathrm{tgt}}$.

Proof. The proof is similar to the proof of Thm. 4.23. Instead of RMW followed by assume, to construct the appropriate context we let $C_i \triangleq x := v$ for the case that $\alpha_i = \bar{\mathtt{W}}(x, v)$. Then, when we show that $\langle s_{i-1}, \theta, \bar{c_i} \rangle \xrightarrow{\mathcal{R}}{}^{*} \langle s_{i-1}, \theta, \alpha_i \rangle$ for the case that $\alpha_i = \bar{\mathtt{W}}(x, v)$, we have that $\langle s_{i-1}, \theta, c_i \rangle \in \lfloor C_i \rfloor$ implies that there exists environment chronicles e', e such that $c_i = e' \cdot \mathtt{W}(x, v) \cdot e$ (but unlike the corresponding case in the proof of Thm. 4.23, we do not necessarily have $e'(s_{i-1})(x) = s_{i-1}(x)$). Then, since we also have $c_i(s_{i-1}) = s_i$ (due to the use of snapshots), using $\mathsf{r}_2^{\not{\mathsf{rmw}}}$, we can rewrite $\langle s_{i-1}, \theta, \bar{c_i} \rangle$ into $\langle s_{i-1}, \theta, \alpha_i \rangle$. □

A version that uses repeated reads instead of snapshots is proved by combining the proofs of Thm. 4.27 and Thm. 5.4. We write $C_{\mathrm{src}} \rightsquigarrow_{\not{\mathsf{rmw}},\not{\mathsf{snapshot}}} C_{\mathrm{tgt}}$, if $\langle P[C_{\mathrm{tgt}}], s \rangle \downarrow s'$ implies $\langle P[C_{\mathrm{src}}], s \rangle \downarrow s'$. for every rmw-free and snapshot-free context P such that $P[C_{\mathrm{src}}]$ and $P[C_{\mathrm{tgt}}]$ are closed.

Theorem 5.5. *If* C_{src} *is loop-free and* $[\![C_{\mathrm{tgt}}]\!]_{\not{\mathsf{rmw}}} \not\subseteq [\![C_{\mathrm{src}}]\!]_{\not{\mathsf{rmw}}}$, *then* $C_{\mathrm{src}} \not\rightsquigarrow_{\not{\mathsf{rmw}},\not{\mathsf{snapshot}}} C_{\mathrm{tgt}}$.

6 RELATED AND FUTURE WORK

We have already discussed the seminal work of **?**, from which we took a lot of inspiration. Our traces consist of write actions, rather than transitions (pairs of states) as in Brookes's traces. This choice has several advantages. First, it directly reflects the property of the operational semantics that each transition updates at most one variable. Second, since reads are not recorded in traces, our concrete semantics, i.e., before imposing any closures, already validates a large variety of refinements, including all those that do not involve writes. In contrast, in Brookes's traces reads are tracked as stuttering transitions, and closures are needed also for refinements of reads (and of `skip`). Third, explicit environment writes in traces allows us have a rule like r_2 that mimics operational simulation that attaches component actions to *one* environment write.

Brookes's traces, which are very similar to the traces used for arguing about soundness of rely/guarantee reasoning [**??**], have provided a useful intuition and formal basis for multiple later frameworks, e.g., [**?????**], which propose relational program logics for reasoning about refinements. Some of these works address the challenge of validating contextual refinements that are conditioned by some assumptions on the concurrent context. Our results on snapshot-free contexts go in this direction, but there is, of course, a variety of more fine-grained assumptions that will allow deriving

useful refinements. For example, [??] used Brookes's semantics for deriving a refinement calculus allowing one to develop full concurrent programs by repeatedly refining a specification. [??] developed a framework for establishing contextual refinement that handles assumptions such as data-race-freedom and data encapsulation in concurrent objects, and demonstrate that their technique is sufficiently expressive for verifying a complex garbage collector. More recently, ?? studied refinements conditioned by separation logic premises. All these works provide sound techniques, whereas our focus is on full abstraction. We hope that our denotational semantics will form a basis for similar continuations.

Another line of work, see e.g., [??], attempts to capture shared-memory concurrency in general, and Brookes's semantics in particular, using monadic constructions following [?], or even as an algebraic theory [??]. A prominent advantage of these approaches is their ability to capture higher-order programs, while we are limited to first-order programs. Additionally, this approach detaches structural refinements from effectful ones and paves the way to type-and-effect systems, enabling reasoning about refinements using assumptions from a type analysis (see e.g., [??]).

Our work handles shared variables admitting sequentially consistent semantics (SC). [?] modified Brookes's semantics to apply for x86-TSO memory (see [?]), and achieved full abstraction using `await` instructions. A large body of work, e.g., [??????], has been devoted to the study of compositional semantics for a weakly consistent memory system that is not necessarily accompanying an existing operational semantics like in our case. A prominent idea there is the use of partially ordered multisets ("pomsets") [?] or event structures [?] that generalize linearly ordered traces, like those we work with. This aligns with axiomatic approaches to concurrency semantics (see, e.g., [?]). However, like operational semantics, axiomatic models are restricted to apply on closed programs.

In the realm of weak memory models, reasoning about correctness of local compiler optimizations is rather challenging and error-prone. Many works have addressed this issue in different levels of formality, e.g., [??????]. Interestingly, it is not always the case that a weaker memory model allows more optimizations than a stronger one (see, e.g., [?]). For instance, weak memory model usually do not support "store-after-load elimination" and "redundant FAA elimination" that are valid under SC (see Fig. 5). Attempting to allow local proofs of optimizations, some of these works develop compositional semantics, but these are restricted to top-level parallel composition. An noteworthy exception in this landscape is the work of ? who developed a denotational semantics for the Release/Acquire weak memory model (see [??]). Their semantics is based on RA's axiomatic formulation, which they generalize to allow "block-local execution graphs" that iterate over all possible context execution graphs, and thus achieving full abstraction. Their blocks are, however, restricted to be sequential programs, which enables local validation of program transformations without actually showing that $[\![C_1 \parallel C_2]\!]$ is a function of $[\![C_1]\!]$ and $[\![C_2]\!]$.

Finally, our notion of contextual refinement is based on partial correctness, and is insensitive to termination. In concurrent programs termination is interesting assuming fairness of the scheduler [?], and, termination is, in fact, generalized into a family of progress conditions [?]. By using infinite traces, Brookes's semantics generalizes to fair infinite runs [?, §10], and is shown to be fully abstract w.r.t. operational "state-trace behaviors" consisting of sequences of states occurring during the computation. We leave the task of incorporating this dimension into our semantics for future work.

A PROOFS

In this appendix we provide proof outlines that were not included in the main text or in our Coq development.

A.1 Proof for Example 4.28

This section is devoted to the missing proof in Example 4.28: $C_{\mathrm{src}} \rightsquigarrow_{\not\text{snapshot}} C_{\mathrm{tgt}}$ for

$$C_{\mathrm{src}} = \texttt{while} * \texttt{do}\ (\texttt{y} := 0\ ;\ \texttt{x} := *\ ;\ \texttt{x} := 0\ ;\ \texttt{y} := *)$$
$$C_{\mathrm{tgt}} = \texttt{y} := 0\ ;\ \texttt{x} := 1\ ;\ \texttt{y} := 1\ ;\ \texttt{x} := 0$$

For this proof we provide a simulation relation that relates operational executions of $P[C_{\mathrm{tgt}}]$ to those of $P[C_{\mathrm{src}}]$.

We start by a simple observation on the operational semantics:

Lemma A.1. *If $\langle C, s\rangle \xrightarrow{\gamma} \langle C', s'\rangle$ and either C is snapshot-free or $\gamma \neq \varepsilon$, then there exists some $x \in \mathrm{Var}$ such that for every s_1 with $s_1(x) = s(x)$, we have $\langle C, s_1\rangle \xrightarrow{\gamma} \langle C', s_1[x \mapsto s'(x)]\rangle$.*

Next, *generalized contexts* are defined by the following grammar:

$$\begin{aligned}\mathbb{P} ::= {} & \texttt{skip} \mid x := E \mid x := * \mid \texttt{assume}(E) \mid \texttt{let}\ a = L\ \texttt{in}\ P \mid C\,;P \mid C \oplus P \mid P \oplus C \mid \\ & \texttt{if}\ E\ \texttt{then}\ P\ \texttt{else}\ C \mid \texttt{if}\ E\ \texttt{then}\ C\ \texttt{else}\ P \mid \texttt{while}\ x\ \texttt{do}\ P \mid \texttt{while} * \texttt{do}\ P \mid \\ & \bowtie \mid \mathbb{P}\,;P \mid C \parallel \mathbb{P} \mid \mathbb{P} \parallel C\end{aligned}$$

We write $\mathbb{P}[C](C_{\bowtie})$ for the command obtained from $\mathbb{P}$ by plugging in C in (all occurrences of) $-$ and $C_{\bowtie}$ in $\bowtie$ (if it exists). A generalized context $\mathbb{P}$ is *peeled* if $\bowtie$ appears in $\mathbb{P}$. We denote by Gctx the set of all generalized contexts, any by PGctx the set of all peeled generalized contexts.

Lemma A.2. *For every context P, there exists a generalized context $\mathbb{P}$ such that $\mathbb{P}[C](C) = P[C]$ for every C.*

Lemma A.3. *Let $\mathbb{P}\{\texttt{skip}/\bowtie\}$ denote the (non-peeled) generalized context obtained from a a generalized context P by substituting $\texttt{skip}$ for $\bowtie$. Then, $\mathbb{P}[C_1](\texttt{skip}) = \mathbb{P}\{\texttt{skip}/\bowtie\}[C_1](C_2)$ for every commands C_1, C_2.*

Lemma A.4. *If $\langle C_2, s\rangle \rightarrow^* \langle C_2', s'\rangle$ and $\mathbb{P}$ is peeled, then $\langle \mathbb{P}[C_1](C_2), s\rangle \rightarrow^* \langle \mathbb{P}[C_1](C_2'), s'\rangle$.*

Lemma A.5. *If $\langle \mathbb{P}[C_1](C_2), s\rangle \rightarrow \langle C', s'\rangle$ and $C_2 \neq \texttt{skip}$, then one of the following holds:*

- *$\mathbb{P}$ is peeled and there exists C_2' such that $C' = \mathbb{P}[C_1](C_2')$ and $\langle C_2, s\rangle \rightarrow \langle C_2', s'\rangle$; or*
- *There exists $\mathbb{P}' \in \mathrm{Gctx}$ such that $C' = \mathbb{P}'[C_1](C_2)$ and the following hold:*
  - *For every $s_1, s_1' \in \mathrm{State}$ and commands C_3, C_4, if $\langle \mathbb{P}[C_1](C_2), s_1\rangle \rightarrow \langle \mathbb{P}'[C_1](C_2), s_1'\rangle$, then $\langle \mathbb{P}[C_3](C_4), s_1\rangle \rightarrow \langle \mathbb{P}'[C_3](C_4), s_1'\rangle$.*
  - *If $\mathbb{P}$ is peeled, then $\mathbb{P}'$ is also peeled.*

Next, for the simulation relation, we define five relations on configurations:

$$\triangleright_1 \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](C_{\mathrm{tgt}}), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](C_{\mathrm{src}}), s\rangle\rangle \mid \mathbb{P} \in \mathrm{Gctx}, s \in \mathrm{State}\}$$
$$\triangleright_2 \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](\texttt{x} := 1\ ;\ \texttt{y} := 1\ ;\ \texttt{x} := 0), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](\texttt{x} := *\ ;\ \texttt{x} := 0\ ;\ \texttt{y} := *\ ;\ C_{\mathrm{src}}), s\rangle\rangle \mid \mathbb{P} \in \mathrm{PGctx}, s \in \mathrm{State}\}$$
$$\triangleright_3 \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](\texttt{y} := 1\ ;\ \texttt{x} := 0), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](\texttt{x} := 0\ ;\ \texttt{y} := *\ ;\ C_{\mathrm{src}}), s\rangle\rangle \mid \mathbb{P} \in \mathrm{PGctx}, s \in \mathrm{State}\}$$
$$\triangleright_4 \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](\texttt{x} := 0), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](C_{\mathrm{src}}), s[\texttt{x} \mapsto 0]\rangle\rangle \mid \mathbb{P} \in \mathrm{PGctx}, s \in \mathrm{State}\}$$
$$\triangleright_5 \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](\texttt{x} := 0), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](\texttt{x} := 0\ ;\ \texttt{y} := *\ ;\ C_{\mathrm{src}}), s[\texttt{y} \mapsto 0]\rangle\rangle \mid \mathbb{P} \in \mathrm{PGctx}, s \in \mathrm{State}\}$$

We let $\triangleright \triangleq \triangleright_1 \cup \triangleright_2 \cup \triangleright_3 \cup \triangleright_4 \cup \triangleright_5$.

Lemma A.6. *If* $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$ *and* $\langle C_1, s_1\rangle \rhd_1 \langle C_2, s_2\rangle$, *then there exists* $\langle C_2', s_2'\rangle$ *such that* $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ *and* $\langle C_1', s_1'\rangle (\rhd_1 \cup \rhd_2) \langle C_2', s_2'\rangle$.

Proof. By definition of $\rhd_1$, we have $C_1 = \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}})$, $C_2 = \mathbb{P}[C_{\text{src}}](C_{\text{src}})$, and $s_1 = s_2$. We apply Lemma A.5 to the step $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$.

If the first case holds, $\mathbb{P}$ is peeled, $C_1' = \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 1\,;\mathsf{y} := 1\,;\mathsf{x} := 0)$, and $s_1' = s_1[\mathsf{y} \mapsto 0]$. We take $C_2' = \mathbb{P}[C_{\text{src}}](C)$ where $C = \mathsf{x} := *\,;\mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}}$, and $s_2' = s_1'$. Since $\langle C_{\text{src}}, s\rangle \longrightarrow^* \langle C, s_2'\rangle$, by Lemma A.4, it follows that $\langle \mathbb{P}[C_{\text{src}}](C_{\text{src}}), s_1\rangle \longrightarrow^* \langle \mathbb{P}[C_{\text{src}}](C), s_2'\rangle$. Then, $\langle C_1', s_1'\rangle \rhd_2 \langle C_2', s_2'\rangle$ follows by definition.

If the second case holds, we take $C_2' = \mathbb{P}'[C_{\text{src}}](C_{\text{src}})$ and $s_2' = s_1'$. Then, $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ and $\langle C_1', s_1'\rangle \rhd_1 \langle C_2', s_2'\rangle$ follow. □

Lemma A.7. *If* $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$ *and* $\langle C_1, s_1\rangle \rhd_2 \langle C_2, s_2\rangle$, *then there exists* $\langle C_2', s_2'\rangle$ *such that* $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ *and* $\langle C_1', s_1'\rangle (\rhd_2 \cup \rhd_3) \langle C_2', s_2'\rangle$.

Proof. Similar to Lemma A.6. □

Lemma A.8. *If* $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$ *and* $\langle C_1, s_1\rangle \rhd_3 \langle C_2, s_2\rangle$, *then there exists* $\langle C_2', s_2'\rangle$ *such that* $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ *and* $\langle C_1', s_1'\rangle (\rhd_3 \cup \rhd_4) \langle C_2', s_2'\rangle$.

Proof. Similar to Lemma A.6. □

Lemma A.9. *If* $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$ *and* $\langle C_1, s_1\rangle \rhd_4 \langle C_2, s_2\rangle$, *then there exists* $\langle C_2', s_2'\rangle$ *such that* $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ *and* $\langle C_1', s_1'\rangle (\rhd_4 \cup \rhd_5 \cup \rhd_1) \langle C_2', s_2'\rangle$.

Proof. By definition of $\rhd_4$, we have $C_1 = \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0)$ and $C_2 = \mathbb{P}[C_{\text{src}}](C_{\text{src}})$, where $\mathbb{P}$ is peeled, as well as $s_2 = s_1[\mathsf{x} \mapsto 0]$. We apply Lemma A.5 to the step $\langle C_1, s_1\rangle \longrightarrow \langle C_1', s_1'\rangle$.

If the first case holds, then $C_1' = \mathbb{P}[C_{\text{tgt}}](\mathtt{skip})$ and $s_1' = s_1[\mathsf{x} \mapsto 0]$. We take $C_2' = \mathbb{P}[C_{\text{src}}](\mathtt{skip})$ and $s_2' = s_1'$. Then, $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$ follows by Lemma A.4 since we have $\langle C_{\text{src}}, s_2\rangle \longrightarrow^* \langle \mathtt{skip}, s_2'\rangle$. To see that $\langle C_1', s_1'\rangle \rhd_1 \langle C_2', s_2'\rangle$ holds, it suffices to note that by Lemma A.3, $C_1' = \mathbb{P}[C_{\text{tgt}}](\mathtt{skip}) = \mathbb{P}'[C_{\text{tgt}}](C_{\text{tgt}})$ and $C_2' = \mathbb{P}[C_{\text{src}}](\mathtt{skip}) = \mathbb{P}'[C_{\text{src}}](C_{\text{src}})$ for $\mathbb{P}' = \mathbb{P}\{\mathtt{skip}/\bowtie\}$.

If the second case holds, then there exists peeled $\mathbb{P}' \in \text{Gctx}$ such that the following hold:

- $C_1' = \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0)$.
- For every s_3, s_3' and commands C_3, C_4, if $\langle \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0), s_3\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0), s_3'\rangle$, then $\langle \mathbb{P}[C_3](C_4), s_3\rangle \longrightarrow \langle \mathbb{P}'[C_3](C_4), s_3'\rangle$.

By Lemma A.1 (applied on $\langle \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0), s_1\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0), s_1'\rangle$), there exists some $z \in \text{Var}$ such that $\langle \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0), s_3\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0), s_3[z \mapsto s_1'(z)]\rangle$ for every s_3 with $s_3(z) = s_1(z)$. Consider two cases:

- Suppose first that $z \neq \mathsf{x}$. We take $C_2' = \mathbb{P}'[C_{\text{src}}](C_{\text{src}})$ and $s_2' = s_1'[\mathsf{x} \mapsto 0] = s_2[z \mapsto s_1'(z)]$. Then, we clearly have $\langle C_1', s_1'\rangle \rhd_4 \langle C_2', s_2'\rangle$. Taking $s_3 = s_2$, we obtain $\langle \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0), s_2\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0), s_2'\rangle$, and so $\langle \mathbb{P}[C_{\text{src}}](C_{\text{src}}), s_2\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{src}}](C_{\text{src}}), s_2'\rangle$.
- Suppose that $z = \mathsf{x}$. We take $C_2' = \mathbb{P}'[C_{\text{src}}](\mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}})$ and $s_2' = s_1'[\mathsf{y} \mapsto 0]$. Then, we clearly have $\langle C_1', s_1'\rangle \rhd_5 \langle C_2', s_2'\rangle$. Taking $s_3 = s_1[\mathsf{y} \mapsto 0]$, we obtain

$$\langle \mathbb{P}[C_{\text{tgt}}](\mathsf{x} := 0), s_1[\mathsf{y} \mapsto 0]\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{tgt}}](\mathsf{x} := 0), s_1'[\mathsf{y} \mapsto 0]\rangle,$$

  and so

$$\langle \mathbb{P}[C_{\text{src}}](\mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}}), s_1[\mathsf{y} \mapsto 0]\rangle \longrightarrow \langle \mathbb{P}'[C_{\text{src}}](\mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}}), s_1'[\mathsf{y} \mapsto 0]\rangle.$$

  Since $\langle C_{\text{src}}, s_1[\mathsf{x} \mapsto 0]\rangle \longrightarrow^* \langle \mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}}, s_1[\mathsf{y} \mapsto 0]\rangle$, by Lemma A.4, it follows that $\langle \mathbb{P}[C_{\text{src}}](C_{\text{src}}), s_1[\mathsf{x} \mapsto 0]\rangle \longrightarrow^* \langle \mathbb{P}[C_{\text{src}}](\mathsf{x} := 0\,;\mathsf{y} := *\,; C_{\text{src}}), s_1[\mathsf{y} \mapsto 0]\rangle$. Together, we obtain that $\langle C_2, s_2\rangle \longrightarrow^* \langle C_2', s_2'\rangle$. □

Lemma A.10. *If* $\langle C_1, s_1\rangle \to \langle C_1', s_1'\rangle$ *and* $\langle C_1, s_1\rangle \rhd_5 \langle C_2, s_2\rangle$, *then there exists* $\langle C_2', s_2'\rangle$ *such that* $\langle C_2, s_2\rangle \to^* \langle C_2', s_2'\rangle$ *and* $\langle C_1', s_1'\rangle(\rhd_4 \cup \rhd_5 \cup \rhd_1)\langle C_2', s_2'\rangle$.

Proof. Similar to Lemma A.9. □

Lemma A.11. $\langle \mathtt{skip}, s\rangle \rhd \langle C, s'\rangle$ *implies that* $C = \mathtt{skip}$ *and* $s = s'$.

Proof. We observe that $\langle \mathtt{skip}, s\rangle \rhd \langle C, s'\rangle$ can only match $\rhd_1$ with $\mathbb{P} = \mathtt{skip}$. By definition of $\rhd_1$, we have $C = \mathbb{P}[C_{\text{src}}](C_{\text{src}}) = \mathtt{skip}$ and $s = s'$. □

Finally, suppose that $\langle P[C_{\text{tgt}}], s\rangle \to^* \langle \mathtt{skip}, s'\rangle$. By Lemma A.2 and definition of $\rhd_1$, there exists $\mathbb{P} \in \text{Gctx}$ such that $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s\rangle \to^* \langle \mathtt{skip}, s'\rangle$ and $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s\rangle \rhd_1 \langle \mathbb{P}[C_{\text{src}}](C_{\text{src}}), s\rangle$. Then, using Lemmas A.6 to A.10, this implies there exists $\langle C_1, s_1\rangle$ such that $\langle P[C_{\text{tgt}}], s\rangle \to^* \langle C_1, s_1\rangle$ and $\langle \mathtt{skip}, s'\rangle \rhd \langle C_1, s_1\rangle$. By Lemma A.11, we conclude that $\langle C_1, s_1\rangle = \langle \mathtt{skip}, s'\rangle$, and so $\langle P[C_{\text{src}}], s\rangle \to^* \langle \mathtt{skip}, s'\rangle$.

A.2 Proof of Lemma 4.26

We denote $\mathcal{R}' \triangleq \{\xrightarrow{\mathrm{r}_1}, \xrightarrow{\mathrm{r}_2}, \xrightarrow{\mathrm{r}_3'}, \xrightarrow{\mathrm{r}_4}\}$.

Proposition A.12. *If* $t \xrightarrow{\mathcal{R}'} t'$, *then* $t = t_1 ; t_2 ; t_3$, $t' = t_1 ; t_2' ; t_3$, *and* $t_2 \xrightarrow{\mathcal{R}^{\text{alt}}} t_2'$ *for some* t_1, t_2, t_3, t_2'.

Proposition A.13. *If* $t_2 \xrightarrow{\mathcal{R}^{\text{alt}}} t_2'$ *and* $t_1 ; t_2 ; t_3$ *is defined, then* $t_1 ; t_2 ; t_3 \xrightarrow{\mathcal{R}'} t_1 ; t_2' ; t_3$.

Proposition A.14. $\xrightarrow{\mathcal{R}^{\text{alt}}} \subseteq \xrightarrow{\mathcal{R}'}$.

Proposition A.15. *If* $t_1 \rightrightarrows t_1'$, $t_2 \rightrightarrows t_2'$ *and* $t_1 ; t_2$ *is defined, then* $t_1 ; t_2 \rightrightarrows t_1' ; t_2'$.

The $\subseteq$ inclusion of Lemma 4.26 is established next.

Proposition A.16. $\rightrightarrows \subseteq \xrightarrow{\mathcal{R}'}{}^*$.

Proof. Suppose that $t \rightrightarrows t'$. The proof proceeds by induction on the proof of $t \rightrightarrows t'$. In the base case, we have $t = t'$, so we are done.

For the induction step, we have $t = t_1 ; t_2$ and $t' = t_1' ; t_2'$ such that $t_1 \xrightarrow{\mathcal{R}^{\text{alt}}}{}^? t_1'$ and $t_2 \rightrightarrows t_2'$. By the induction hypothesis we have $t_2 \xrightarrow{\mathcal{R}'}{}^* t_2'$. By Prop. 4.19, as $t_1' ; t_2'$ is defined, $t_1 ; t_2'$ is also defined. So we can apply Prop. 4.12 to obtain $t_1 ; t_2 \xrightarrow{\mathcal{R}'}{}^* t_1 ; t_2'$. Next, $t_1 ; t_2' \xrightarrow{\mathcal{R}'}{}^? t_1' ; t_2'$ follows by Prop. A.13. We conclude that $t_1 ; t_2 \xrightarrow{\mathcal{R}'}{}^* t_1' ; t_2'$. □

Proposition A.17. $\xrightarrow{\mathcal{R}'} \subseteq \rightrightarrows$.

Proof. Suppose that $\langle s, \theta, c\rangle = t \xrightarrow{\mathcal{R}'} t'$. By Prop. A.12, we have $t = t_1 ; t_2 ; t_3$, $t' = t_1 ; t_2' ; t_3$, and $t_2 \xrightarrow{\mathcal{R}^{\text{alt}}} t_2'$ for some t_1, t_2, t_3, t_2'. We derive $t \rightrightarrows t'$ as follows:

$$\dfrac{t_1 \xrightarrow{\mathcal{R}^{\text{alt}}}{}^? t_1 \qquad \dfrac{t_2 \xrightarrow{\mathcal{R}^{\text{alt}}}{}^? t_2' \qquad \dfrac{t_3 \xrightarrow{\mathcal{R}^{\text{alt}}}{}^? t_3 \qquad \overline{\langle c(s), \theta, \varepsilon\rangle \rightrightarrows \langle c(s), \theta, \varepsilon\rangle}}{t_3 \rightrightarrows t_3}}{t_2 ; t_3 \rightrightarrows t_2' ; t_3}}{t_1 ; t_2 ; t_3 \rightrightarrows t_1 ; t_2' ; t_3}$$

□

The $\supseteq$ inclusion of Lemma 4.26, i.e., that $t \xrightarrow{\mathcal{R}'}{}^{*} t'$ implies $t \rightrightarrows t'$, is proved by induction on the number of rewrite steps in $\xrightarrow{\mathcal{R}'}{}^{*}$. Proposition A.17 establishes the base case, and Prop. A.19 gives us the induction step.

Proposition A.18. $\xrightarrow{\mathcal{R}'} ; \xrightarrow{\mathcal{R}^{\text{alt}}} \subseteq \rightrightarrows$.

Proof. We analyze the possible cases of the rules used in both rewrites. First, we demonstrate here the case of $t \xrightarrow{\text{r}_1} t' \xrightarrow{\text{r}_1^{\text{alt}}} t''$. In this case we have:

$$t = \langle s, \theta, c_1 \cdot m_1 \cdot \mathtt{W}(x, v) \cdot m_2 \cdot c_2 \rangle \xrightarrow{\text{r}_1} \langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2 \rangle = t' = \langle s, \theta, m_1' \cdot \mathtt{W}(x', v') \cdot m_2' \rangle \xrightarrow{\text{r}_1^{\text{alt}}} \langle s, \theta, \mathtt{W}(x', v') \rangle = t''$$

where $m_1, m_2 \in \text{CmpChro}$, $(m_1 \cdot \mathtt{W}(x, v) \cdot m_2)(c_1(s)) = c_1(s)[x \mapsto v]$, $m_1', m_2' \in \text{CmpChro}$ and $(m_1' \cdot \mathtt{W}(x', v') \cdot m_2')(s) = s[x' \mapsto v']$. It follows that c_1, c_2 are component chronicles, and $(c_1 \cdot m_1 \cdot \mathtt{W}(x, v) \cdot m_2 \cdot c_2)(s) = (c_1 \cdot \mathtt{W}(x, v) \cdot c_2)(s) = (m_1' \cdot \mathtt{W}(x', v') \cdot m_2')(s) = s[x' \mapsto v']$. Therefore, $t \xrightarrow{\text{r}_1^{\text{alt}}} t''$. By Propositions A.14 and A.17, it follows that $t \rightrightarrows t''$.

We demonstrate also the case of $t \xrightarrow{\text{r}_4} t' \xrightarrow{\text{r}_1^{\text{alt}}} t''$. In this case we have:

$$t = \langle s, \theta, c_1 \cdot c_2 \rangle \xrightarrow{\text{r}_4} \langle s, \theta, c_1 \cdot \mathtt{W}(x, v) \cdot c_2 \rangle = t' = \langle s, \theta, m_1' \cdot \mathtt{W}(x', v') \cdot m_2' \rangle \xrightarrow{\text{r}_1^{\text{alt}}} \langle s, \theta, \mathtt{W}(x', v') \rangle = t''$$

where $c_1(s)(x) = v$, $m_1', m_2' \in \text{CmpChro}$ and $(m_1' \cdot \mathtt{W}(x', v') \cdot m_2')(s) = s[x' \mapsto v']$. It follows that c_1, c_2 are component chronicles. We also observe that $(c_1 \cdot c_2)(s) = (c_1 \cdot \mathtt{W}(x, v) \cdot c_2)(s) = (m_1' \cdot \mathtt{W}(x', v') \cdot m_2')(s) = s[x' \mapsto v']$. Now, if $\mathtt{W}(x', v')$ is present inside $c_1 \cdot c_2$, we can rewrite $\langle s, \theta, c_1 \cdot c_2 \rangle$ into $\langle s, \theta, \mathtt{W}(x', v') \rangle$ using r_1^{alt}. Otherwise, $s(x') = v'$, and we derive $t \rightrightarrows t''$ as follows:

$$\dfrac{\langle s, \theta, \varepsilon \rangle \xrightarrow{\text{r}_4^{\text{alt}}} \langle s, \theta, \mathtt{W}(x', v') \rangle \qquad \dfrac{\langle s, \theta, c_1 \cdot c_2 \rangle \xrightarrow{\text{r}_3^{\text{alt}}} \langle s, \theta, \varepsilon \rangle \qquad \overline{\langle s, \theta, \varepsilon \rangle \rightrightarrows \langle s, \theta, \varepsilon \rangle}}{\langle s, \theta, c_1 \cdot c_2 \rangle \rightrightarrows \langle s, \theta, \varepsilon \rangle}}{\langle s, \theta, c_1 \cdot c_2 \rangle \rightrightarrows \langle s, \theta, \mathtt{W}(x', v') \rangle}$$

□

Proposition A.19. $\xrightarrow{\mathcal{R}'} ; \rightrightarrows \subseteq \rightrightarrows$.

Proof. Suppose that $t \xrightarrow{\mathcal{R}'} t' \rightrightarrows t''$. We prove that $t \rightrightarrows t''$ by induction on $|t| + |t''|$ (where $|t_0|$ is defined as the length of the chronicle of t_0 for every trace t_0). In the base case, we have $|t| + |t''| = 0$, which implies $t = t'' = \langle s, \theta, \varepsilon \rangle$ (by Prop. 4.19) for some state s and store θ, and we are done.

For the induction step, we have $|t| + |t''| > 0$. First, if $t' = t''$, then by Prop. A.17, $t \xrightarrow{\mathcal{R}'} t'$ implies $t \rightrightarrows t' = t''$, and we are done. Otherwise, we have $t' = t_1' ; t_2'$ and $t'' = t_1'' ; t_2''$ such that $t_1' \xrightarrow{\mathcal{R}^{\text{alt}}}{}^{?} t_1''$ and $t_2' \rightrightarrows t_2''$. We may also assume that $|t_1'| + |t_1''| \geq 1$. By Prop. A.12, we have $t = u_1 ; u_2 ; u_3$ and $t' = u_1 ; u_2' ; u_3$, such that $u_2 \xrightarrow{\mathcal{R}^{\text{alt}}} u_2'$.

We observe that $|u_2'| \leq 1$, so u_2' is contained in either t_1' or t_2'. Consider the two cases:

- u_2' is contained in t_1': In this case, there exists u_3' such that $u_3 = u_3' ; t_2'$. By Prop. A.13, we have $(u_1 ; u_2 ; u_3') \xrightarrow{\mathcal{R}'} (u_1 ; u_2' ; u_3') = t_1' \xrightarrow{\mathcal{R}^{\text{alt}}}{}^{?} t_1''$. Consider two cases:
  - $t_1' \neq t_1''$: Then, by Prop. A.18 we have $(u_1 ; u_2 ; u_3') \rightrightarrows t_1''$. Since we have $t = (u_1 ; u_2 ; u_3') ; t_2'$, by Prop. A.15, it follows that $t \rightrightarrows t''$.

- $t_1' = t_1''$: Then, we have $u_1 = u_1$, $u_2 \xrightarrow{\mathcal{R}^{\text{alt}}} u_2'$ and $u_3' = u_3'$. Together with $t_2' \rightrightarrows t_2''$, it follows that $t \rightrightarrows t''$.

- u_2' is contained in t_2': In this case, there exists u_1' such that $u_1 = t_1' ; u_1'$. By Prop. A.13, we have $(u_1' ; u_2 ; u_3) \xrightarrow{\mathcal{R}'} (u_1' ; u_2' ; u_3) = t_2' \rightrightarrows t_2''$. We have $|u_1' ; u_2 ; u_3| + |t_2''| < |t'| + |t''|$ (the difference is $|t_1'| + |t_1''|$), and by the induction hypothesis we obtain $(u_1' ; u_2 ; u_3) \rightrightarrows t_2''$. Together with $t_1' \xrightarrow{\mathcal{R}^{\text{alt}}{}^{?}} t_1''$, it follows that $t \rightrightarrows t''$. □

Proposition A.20. *If $t \rightrightarrows (t_1' ; t_2')$, then there exist t_1, t_2 such that $t = t_1 ; t_2$, $t_1 \rightrightarrows t_1'$, $t_2 \rightrightarrows t_2'$.*

Proof. We consider the steps in proof of $t \rightrightarrows t'$, which contain rewrites of the form $u_i \xrightarrow{\mathcal{R}^{\text{alt}}{}^{?}} u_i'$, and put them in one sequence until we have $|u_1' ; u_2'; \dots ; u_n'| = |t_1'|$. If we cannot take the next step, as we have $|u_1' ; u_2'; \dots ; u_n'| < |t_1'|$ and $|u_1' ; u_2'; \dots ; u_{n+1}'| > |t_1'|$, we observe that this can only happen if $|u_{n+1}'| > 1$. This implies no rule from $\mathcal{R}^{\text{alt}}$ was applied, and $u_{n+1} = u_{n+1}'$. So we can split u_{n+1} into two parts to satisfy the length condition.

It follows that $(u_1 ; u_2; \dots ; t_n) = t_1 \rightrightarrows t_1'$, and similar for other claims. □

B CONTEXT SEMANTICS

Definition B.1. A transformation from a context P_{src} to a context P_{tgt} is *sound*, denoted by $P_{\text{src}} \rightsquigarrow P_{\text{tgt}}$, if $P_{\text{src}}[C] \rightsquigarrow P_{\text{tgt}}[C]$ for every command C. **O**: *missing "closedness" condition?* **M**: *no, it is required for $P[P_{\text{tgt}}[C]]$ to be closed*

In Fig. 6 we present an inductive definition of the concrete semantics for contexts. It associates a function $\lfloor P \rfloor : \mathcal{P}(\text{Trace}) \to \mathcal{P}(\text{Trace})$ to every context P.

Proposition B.2. *For every context P and command C, we have $\lfloor P \rfloor(\lfloor C \rfloor) = \lfloor P[C] \rfloor$.*

Proof. The proof proceeds by induction on P. We demonstrate here the case of $P = P' \parallel C_1$. We have $t \in \lfloor P \rfloor(\lfloor C \rfloor)$, if and only if $t = t_1 \parallel t_2$, such that $t_1 \in \lfloor P' \rfloor(\lfloor C \rfloor)$ and $t_2 \in \lfloor C_1 \rfloor$. By induction hypothesis, $\lfloor P' \rfloor \lfloor C \rfloor = \lfloor P'[C] \rfloor$; we also observe that $P'[C] \parallel C_1 = P[C]$. So the last is equivalent to $t_1 \parallel t_2 \in \lfloor P[C] \rfloor$, as required. □

O: *given the last proposition, I am wondering: can we define: $\lfloor P \rfloor = \lambda T. \lfloor P[C_T] \rfloor$ for some C_T such that $\lfloor C_T \rfloor = T$?* **M**: *this requires to "pick" some C_T—otherwise yes, we can* **O**: *Then I don't think we need an inductive definition at all. Perhaps this part is actually a theorem formalizing the comments we have in the text after the compositionality results.*

The adequacy of this semantics follows:

Proposition B.3. *If $\lfloor P_{\text{tgt}} \rfloor(\lfloor C \rfloor) \subseteq \lfloor P_{\text{src}} \rfloor(\lfloor C \rfloor)$ for every command C, then $P_{\text{src}} \rightsquigarrow P_{\text{tgt}}$.*

We also define $[\![P]\!] \triangleq \lambda T. (\lfloor P \rfloor(T))^{\mathcal{R}}$.

Proposition B.4. *For every context P and command C, we have $[\![P]\!](\lfloor C \rfloor) = [\![P[C]]\!]$.*

Proposition B.5. *If $[\![P_{\text{tgt}}]\!](\lfloor C \rfloor) \subseteq [\![P_{\text{src}}]\!](\lfloor C \rfloor)$ for every command C, then $P_{\text{src}} \rightsquigarrow P_{\text{tgt}}$.*

Theorem B.6. *If $[\![P_{\text{tgt}}]\!](\lfloor C \rfloor) \not\subseteq [\![P_{\text{src}}]\!](\lfloor C \rfloor)$ for some command C, then $P_{\text{src}} \not\rightsquigarrow P_{\text{tgt}}$.*

C FAIRNESS

We try to support infinite computations with similar methods.

$$\frac{t \in T}{t \in \lfloor - \rfloor(T)} \qquad \frac{e \in \mathsf{EnvChro} \quad \langle \theta(L), e(s) \rangle \xrightarrow{\gamma} \langle v, s' \rangle \quad \langle s', \theta[a \mapsto v], c \rangle \in \lfloor P \rfloor(T)}{\langle s, \theta, e \cdot \gamma \cdot c \rangle \in \lfloor \mathtt{let}\ a = L\ \mathtt{in}\ P \rfloor(T)} \qquad \frac{t_1 \in \lfloor P \rfloor(T) \quad t_2 \in \lfloor C \rfloor}{t_1 \,;\, t_2 \in \lfloor P \,;\, C \rfloor(T)}$$

$$\frac{t_1 \in \lfloor C \rfloor \quad t_2 \in \lfloor P \rfloor(T)}{t_1 \,;\, t_2 \in \lfloor C \,;\, P \rfloor(T)} \qquad \frac{t_1 \in \lfloor P \rfloor(T) \quad t_2 \in \lfloor C \rfloor}{t_1 \parallel t_2 \in \lfloor P \parallel C \rfloor(T)} \qquad \frac{t_1 \in \lfloor C \rfloor \quad t_2 \in \lfloor P \rfloor(T)}{t_1 \parallel t_2 \in \lfloor C \parallel P \rfloor(T)} \qquad \frac{t \in \lfloor P \rfloor(T) \cup \lfloor C \rfloor}{t \in \lfloor C \oplus P \rfloor(T)}$$

$$\frac{t \in \lfloor P \rfloor(T) \cup \lfloor C \rfloor}{t \in \lfloor P \oplus C \rfloor(T)} \qquad \frac{\theta(E) \neq 0 \implies \langle s, \theta, c \rangle \in \lfloor C \rfloor \quad \theta(E) = 0 \implies \langle s, \theta, c \rangle \in \lfloor P \rfloor(T)}{\langle s, \theta, c \rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ C\ \mathtt{else}\ P \rfloor(T)}$$

$$\frac{\theta(E) = 0 \implies \langle s, \theta, c \rangle \in \lfloor C \rfloor \quad \theta(E) \neq 0 \implies \langle s, \theta, c \rangle \in \lfloor P \rfloor(T)}{\langle s, \theta, c \rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ P\ \mathtt{else}\ C \rfloor(T)} \qquad \frac{\langle e(s), \theta, c \rangle \in \lfloor P \rfloor(T) \quad e(s)(x) \neq 0 \quad t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P \rfloor(T)}{\langle s, \theta, e \cdot c \rangle \,;\, t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P \rfloor(T)}$$

$$\frac{e_1, e_2 \in \mathsf{EnvChro} \quad e_1(s)(x) = 0}{\langle s, \theta, e_1 \cdot e_2 \rangle \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P \rfloor(T)} \qquad \frac{t_1 \in \lfloor P \rfloor(T) \quad t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P \rfloor(T)}{t_1 \,;\, t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P \rfloor(T)}$$

$$\frac{e \in \mathsf{EnvChro}}{\langle s, \theta, e \rangle \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P \rfloor(T)}$$

Fig. 6. Concrete Context Semantics: $t \in \lfloor P \rfloor(T)$

C.1 Operational Semantics

Intuitively, an infinite computation of a command C is *fair* if every forever active thread in C executes infinitely many times.

Formally, for each thread (which can later be divided into more threads, by parallel composition) we introduce a natural number ("fuel"), which is a counter of how many times this thread can execute before giving the initiative to other threads.

The structure of all threads that are active at some point of execution is represented by a binary tree whose nodes are labeled by natural numbers.

Definition C.1. A *fuel tree* T is a binary tree whose nodes are labeled by natural numbers (formally, elements generated by the grammar $T ::= \langle n \rangle \mid \langle n, T, T \rangle$). A partial order $<$ on trees is defined by $T < T'$ if one of the following holds:

- $T = \langle n \rangle$ and $T' = \langle n' \rangle$ for some $n < n'$; or
- $T = \langle n, T_1, T_2 \rangle$ and $T' = \langle n', T_1, T_2 \rangle$ for some $n < n'$ and trees T_1, T_2.

We use the notation $T[n]$ to set the label of T's root equal to n.

The operational semantics is defined using a relation $\rightarrow$ on configurations of the form $\langle C, s, T \rangle$. The relation is given in §C.1.

$$\frac{\forall C_1, C_2.\, C \neq C_1 \parallel C_2 \quad \forall C_1, C_2, C_3.\, C \neq (C_1 \parallel C_2)\,;\, C_3 \quad \langle C, s\rangle \xrightarrow{\gamma} \langle C', s'\rangle}{\langle C, s, \langle n\rangle\rangle \xrightarrow{\gamma} \langle C', s', \langle n\rangle\rangle} \qquad \frac{\langle C_1, s, T\rangle \xrightarrow{\gamma} \langle C_1', s', T'\rangle}{\langle C_1\,;\, C_2, s, T\rangle \xrightarrow{\gamma} \langle C_1'\,;\, C_2, s', T'\rangle} \qquad \frac{T = \langle n, \langle n\rangle, \langle n\rangle\rangle}{\langle C_1 \parallel C_2, s, \langle n\rangle\rangle \xrightarrow{\varepsilon} \langle C_1 \parallel C_2, s, T\rangle}$$

$$\frac{\langle C_1, s, T_1\rangle \xrightarrow{\gamma} \langle C_1', s', T_1'\rangle \quad T = \langle n, T_1, T_2\rangle \quad T_1'' < T_1' \quad T_2 < T_2'' \quad T' = \langle n, T_1'', T_2''\rangle}{\langle C_1 \parallel C_2, s, T\rangle \xrightarrow{\gamma} \langle C_1' \parallel C_2, s', T'\rangle} \qquad \frac{\langle C_2, s, T_2\rangle \xrightarrow{\gamma} \langle C_2', s', T_2'\rangle \quad T = \langle n, T_1, T_2\rangle \quad T_1 < T_1'' \quad T_2'' < T_2' \quad T' = \langle n, T_1'', T_2''\rangle}{\langle C_1 \parallel C_2, s, T\rangle \xrightarrow{\gamma} \langle C_1 \parallel C_2', s', T'\rangle} \qquad \frac{T = \langle n, \_, \_\rangle}{\langle C \parallel \mathtt{skip}, s, T\rangle \xrightarrow{\varepsilon} \langle C, s, \langle n\rangle\rangle}$$

$$\frac{T = \langle n, \_, \_\rangle}{\langle \mathtt{skip} \parallel C, s, T\rangle \xrightarrow{\varepsilon} \langle C, s, \langle n\rangle\rangle}$$

Fig. 7. Fuel Operational Semantics

$$\frac{\forall C_1, C_2.\, C \neq C_1 \parallel C_2 \quad \forall C_1, C_2, C_3.\, C \neq (C_1 \parallel C_2)\,;\, C_3 \quad \langle C, s\rangle \rightarrow \langle C', s'\rangle}{\langle C, s, \langle \nu\rangle\rangle \xrightarrow{\emptyset} \langle C', s', \langle \nu\rangle\rangle} \qquad \frac{\langle C_1, s, \mathbf{N}\rangle \xrightarrow{N} \langle C_1', s', \mathbf{N}'\rangle}{\langle C_1\,;\, C_2, s, \mathbf{N}\rangle \xrightarrow{N} \langle C_1'\,;\, C_2, s', \mathbf{N}'\rangle} \qquad \frac{\mathbf{N} = \langle \nu, \langle \nu_1\rangle, \langle \nu_2\rangle\rangle}{\langle C_1 \parallel C_2, s, \langle \nu\rangle\rangle \xrightarrow{\emptyset} \langle C_1 \parallel C_2, s, \mathbf{N}\rangle}$$

$$\frac{\langle C_1, s, \mathbf{N}_1\rangle \xrightarrow{N} \langle C_1', s', \mathbf{N}_1'\rangle \quad \mathbf{N} = \langle \nu_0, \mathbf{N}_1, \mathbf{N}_2\rangle \quad \mathbf{N}' = \langle \nu_0, \mathbf{N}_1', \mathbf{N}_2\rangle}{\langle C_1 \parallel C_2, s, \mathbf{N}\rangle \xrightarrow{N \cup \mathrm{NameSet}(\mathbf{N}_2)} \langle C_1' \parallel C_2, s', \mathbf{N}'\rangle} \qquad \frac{\langle C_2, s, \mathbf{N}_2\rangle \xrightarrow{N} \langle C_2', s', \mathbf{N}_2'\rangle \quad \mathbf{N} = \langle \nu_0, \mathbf{N}_1, \mathbf{N}_2\rangle \quad \mathbf{N}' = \langle \nu_0, \mathbf{N}_1, \mathbf{N}_2'\rangle}{\langle C_1 \parallel C_2, s, \mathbf{N}\rangle \xrightarrow{N \cup \mathrm{NameSet}(\mathbf{N}_1)} \langle C_1 \parallel C_2', s', \mathbf{N}'\rangle}$$

$$\frac{\mathbf{N} = \langle \nu, \_, \_\rangle}{\langle C \parallel \mathtt{skip}, s, \mathbf{N}\rangle \xrightarrow{\emptyset} \langle C, s, \langle \nu\rangle\rangle} \qquad \frac{\mathbf{N} = \langle \nu, \_, \_\rangle}{\langle \mathtt{skip} \parallel C, s, \mathbf{N}\rangle \xrightarrow{\emptyset} \langle C, s, \langle \nu\rangle\rangle}$$

Fig. 8. No-Fuel Operational Semantics — **O**: *new version*

Next, we define a similar semantics without fuel.
We assume a infinite set Name of *names*, and use ν as a meta-variable to range over this set.

Definition C.2. A *name tree* $\mathbf{N}$ is a binary tree whose nodes are labeled by names from Name, such that every name occur at most once in $\mathbf{N}$. We write $\mathrm{NameSet}(\mathbf{N})$ to retrieve the set of all names that occur in $\mathbf{N}$.

Fig. 8 presents small-step operational semantics where each step is annotated with a set N which tracks the "inactive names" of that step. Intuitively, infinite runs are fair if no name continuously appears in this set from some point on.

C.2 Fair infinite runs

We write $\langle C, s\rangle \downarrow_{\mathsf{fair}} \infty$ if $\langle C, s, T\rangle \downarrow_{\mathsf{fair}} \infty$ for some fuel tree T, and the last one is given by coinductive definition:

$$\frac{\langle C, s, T\rangle \to \langle C', s', T'\rangle \qquad \langle C', s', T'\rangle \downarrow_{\mathsf{fair}} \infty}{\langle C, s, T\rangle \downarrow_{\mathsf{fair}} \infty}$$

We write $\langle C, s\rangle \downarrow_{\mathsf{fair}} s_1, s_2, \ldots$ if $\langle C, s, T\rangle \downarrow_{\mathsf{fair}} s_1, s_2, \ldots$ for some fuel tree T, and the last one is given by coinductive definition:

$$\frac{\langle C, s, T\rangle \to^* \langle C', s_1, T'\rangle \qquad \langle C', s_1, T'\rangle \downarrow_{\mathsf{fair}} s_2, s_3, \ldots}{\langle C, s, T\rangle \downarrow_{\mathsf{fair}} s_1, s_2, \ldots}$$

For every $\nu, \nu' \in \mathsf{Name}$, we define inductively the properties $\mathrm{G}_\nu, \mathrm{G}^2_{\nu,\nu'}$ of the sequence of name sets:

$$\frac{\nu \notin N_1}{(N_1, N_2, \ldots) \in \mathrm{G}_\nu} \qquad \frac{(N_2, N_3, \ldots) \in \mathrm{G}_\nu}{(N_1, N_2, \ldots) \in \mathrm{G}_\nu} \qquad \frac{(N_1, N_2, \ldots) \in \mathrm{G}_{\nu'} \quad \nu \notin N_1}{(N_1, N_2, \ldots) \in \mathrm{G}^2_{\nu,\nu'}} \qquad \frac{(N_2, N_3, \ldots) \in \mathrm{G}^2_{\nu,\nu'}}{(N_1, N_2, \ldots) \in \mathrm{G}^2_{\nu,\nu'}}$$

We denote $\mathrm{G} = \bigwedge_{\nu \in \mathsf{Name}} \mathrm{G}_\nu$ and $\mathrm{G}^2 = \bigwedge_{\nu,\nu' \in \mathsf{Name}} \mathrm{G}^2_{\nu,\nu'}$.

We give a coinductive definition of $\langle C, s, \mathbf{N}\rangle \downarrow_{\mathsf{fair}} \infty(N_1, N_2, \ldots)$:

$$\frac{\begin{array}{c}\langle C, s, \mathbf{N}\rangle \xrightarrow{N_1} \langle C', s', \mathbf{N}'\rangle \\ \langle C', s', \mathbf{N}'\rangle \downarrow_{\mathsf{fair}} \infty(N_2, N_3, \ldots) \\ \mathrm{G}^2(N_1, N_2, \ldots)\end{array}}{\langle C, s, \mathbf{N}\rangle \downarrow_{\mathsf{fair}} \infty(N_1, N_2, \ldots)}$$

We give a coinductive definition of $\langle C, s\rangle \downarrow s_1, s_2, \ldots$:

$$\frac{\langle C, s\rangle \to^* \langle C', s_1\rangle \qquad \langle C', s_1\rangle \downarrow s_2, s_3, \ldots}{\langle C, s\rangle \downarrow s_1, s_2, \ldots}$$

We write $\langle C, s\rangle \downarrow \infty$ if $\langle C, s\rangle \downarrow s_1, s_2, \ldots$ holds for some infinite sequence of states.

TODO: *add notion for 'print' instructions*

We write $T \cong \mathbf{N}$ if the trees have isomorphic structures, ignoring their labels (natural numbers and names, respectively).

Proposition C.3. $\langle C, s, \mathbf{N}\rangle \downarrow_{\mathsf{fair}} \infty(N_1, N_2, \ldots) \Leftrightarrow \exists T.\, \langle C, s, T\rangle \downarrow_{\mathsf{fair}} \infty \wedge T \cong \mathbf{N}$.

M: *todo: add the chronicle and make the run possibly finite and possibly interrupted*

Proof. First we observe a property of fuel operational semantics: a step in one thread results in decreasing the numbers in the corresponding node and all nodes above it, except the root.

($\Rightarrow$) We define one more property $\mathrm{G}_{\nu,\nu'}$ inductively:

$$\frac{\nu \notin N_1 \vee \nu' \notin N_1}{(N_1, N_2, \ldots) \in \mathrm{G}_\nu \cap \mathrm{G}_{\nu'}} \qquad \frac{(N_2, N_3, \ldots) \in \mathrm{G}'_{\nu,\nu'}}{(N_1, N_2, \ldots) \in \mathrm{G}_{\nu,\nu'}}$$

We define $T(\mathbf{N})$ as follows: for the root, we choose an arbitrary fixed natural number; for every other node ν, we choose the length of the proof of $\mathrm{G}_{\nu'}(N_k, N_{k+1}, \ldots)$ inside the proof of $\mathrm{G}'_{\nu,\nu'}(N_1, N_2, \ldots)$, where ν' is the name of sibling node.

Claim C.3.1: The property $(N_1, N_2, \ldots) \in \mathrm{G}'_{\nu,\nu'}$ holds for every non-root node ν and its sibling ν'.

Proof. Let ν_0 be the least common ancestor of ν, ν'; as we have $(N_1, N_2, \ldots) \in \mathrm{G}_{\nu_0}$, the first clause will hold for some N_k. Next we observe that $(N_1, N_2, \ldots) \in \mathrm{G}^2_{\nu_0,\nu}$ implies $(N_k, N_{k+1}, \ldots) \in \mathrm{G}_\nu$, and similarly for $(N_k, N_{k+1}, \ldots) \in \mathrm{G}_{\nu'}$. □

We prove that, if $\langle C, s, \mathbf{N}\rangle \xrightarrow{N_1} \langle C', s', \mathbf{N}'\rangle$ and the (possibly finite) sequence $(N_1, N_2, \ldots) \in \mathrm{G}^2$, then $\langle C, s, T(\mathbf{N})\rangle \to \langle C', s', T'(\mathbf{N}')\rangle$. (Here $T'()$ denotes the function based on the sequence $(N_2, N_3, \ldots)$).

The proof is done by induction on C. We demonstrate here the case $C = C_1 \parallel C_2$. Suppose $\langle C_1, s, \mathbf{N}_1\rangle \xrightarrow{N'_1} \langle C'_1, s', \mathbf{N}'_1\rangle$; we have $\mathbf{N} = \langle \nu_0, \mathbf{N}_1, \mathbf{N}_2\rangle$, $\mathbf{N}' = \langle \nu_0, \mathbf{N}'_1, \mathbf{N}_2\rangle$, $N_1 = N'_1 \cup \mathsf{NameSet}(\mathbf{N}_2)$.

Let ν_1, ν_2 be the roots of $\mathbf{N}_1, \mathbf{N}_2$, and $n(\nu), n'(\nu)$ be the functions extracted from our definition of $T(\mathbf{N}), T'(\mathbf{N})$. We observe that $\nu_1 \notin N_1, \nu_2 \in N_1$. Suppose k is minimal such that $\nu_2 \notin N_k$. Then $n(\nu_1) = k$, and $n'(\nu_1) < k$ (the next point from which we will observe G_{ν_2} is at least 2 and at most k). We also have $n(\nu_2) = 1$ and $n'(\nu_2) \geq 1$, since it is the minimal possible value.

For any other node $\nu \in \mathbf{N}_2$ (together with its sibling ν'), we have $\nu, \nu' \in N_1$, therefore $n'(\nu) = n(\nu)$.

For any other node $\nu \in \mathbf{N}_1$, we have $\nu \in N_1 \Leftrightarrow \nu \in N'_1$. Moreover, G^2 holds for the subsequence obtained from $(N_1, N_2, \ldots)$ by considering only the steps where $\nu_1 \notin N_i, \nu_2 \in N_i$ (or vice versa) and reducing N_i to N'_i as with N_1. So we have $\langle C_1, s, T(\mathbf{N}_1)\rangle \to \langle C'_1, s', T'(\mathbf{N}'_1)\rangle$ by induction hypothesis. Combining this together, we conclude that $\langle C_1 \parallel C_2, s, T(\mathbf{N})\rangle \to \langle C'_1 \parallel C_2, s', T'(\mathbf{N}')\rangle$.

M: *todo: define the subsequence and check* G^2 *for it*

($\Leftarrow$) We construct the tree $\mathbf{N}$ choosing arbitrary new names for each node; to keep the same names as we had at the step before, we may observe that C can become a finite number of commands during the computation and construct the trees for them beforehand. So if a node is present in both $\mathbf{N}, \mathbf{N}'$ at some step, we guarantee that it has the same name.

Now suppose that G_ν does not hold for some ν_0. In particular, it means $\nu_0 \in \mathsf{NameSet}(\mathbf{N})$ and it is not the root of $\mathbf{N}$. Given a tree, we call an antichain A inside it *proper* if it does not cover all the leaves of the tree.

We note that, if ν is the parent of ν' in the tree, then $\neg\mathrm{G}_\nu$ implies $\neg\mathrm{G}'_\nu$. (*) (It follows from $\nu \in N_1 \Rightarrow \nu' \in N_1$).

Consider all the nodes except the root in $\mathbf{N}$ for which G_ν holds and denote A—the minimal elements of this set.

We prove A cannot be a proper antichain using induction on $|\mathbf{N}|$.

When $|\mathbf{N}| = 1$, our claim is clear as there are no proper antichains.

When $|\mathsf{A}| = 1$, our claim is clear as one thread cannot take steps since its fuel reaches 0. The thread in A is the only one there, so its only parent is the root (follows from (*)), and its sibling does not take steps at all.

For the case of $|\mathsf{A}| > 1$ and $|\mathbf{N}| > 1$, suppose the opposite. Then there exists a thread in A (to be exact, in the corresponding T) that takes infinitely many steps, thus decreasing fuel in its node and all nodes above (except the root). To make this possible, we should recharge fuel in all these nodes infinitely many times. Hence any node, which is sibling to one of these nodes and is the root of some subtree T', should decrease infinitely many times, which is equivalent to some set of threads in T' taking infinitely many steps; we consider

the minimal threads among them forming an antichain. But, as $|T'| < |T|$, by induction hypothesis we know that this couldn't be a proper anichain of threads in T'. So all leaves in T' should be covered by participating alive threads, and so should all the leaves in T (by observing that the root of T' could be any node, sibling to some node above an initial thread). We obtain a contradiction as A is not a proper antichain.
It follows that v_0 has some parent v' for which $G_{v'}$ holds. As v_0 is always in N_i, its parent is always in the tree. So for the infinite (interrupted) run of the command corresponding to v' subtree, we have the same problem as G_{v_0} does not hold. As this tree is smaller, this is a contradiction.
M: *todo: define this subtree run*
M: *todo: prove* G^2*, not* G

□

C.3 Contextual Refinement

We define two notions of termination-sensitive contextual refinements:

- $C_{\text{src}} \rightsquigarrow_{\text{fair}} C_{\text{tgt}}$ if $C_{\text{src}} \rightsquigarrow C_{\text{tgt}}$ and for every context P such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, we have that $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} \infty$ implies $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} \infty$.
- $C_{\text{src}} \rightsquigarrow_{\text{fairseq}} C_{\text{tgt}}$ if $C_{\text{src}} \rightsquigarrow C_{\text{tgt}}$ and for every context P such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, we have that $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, \ldots$ implies $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, \ldots.$

Proposition C.4. *If* $C_{\text{src}} \rightsquigarrow_{\text{fairseq}} C_{\text{tgt}}$*, then* $C_{\text{src}} \rightsquigarrow_{\text{fair}} C_{\text{tgt}}$.

Proof. Suppose $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} \infty$, and let $s_1, s_2, \ldots$ be the infinite sequence of states that this computation visits. We have $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, \ldots$ and $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, \ldots$; therefore $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} \infty$. □

Example C.5. For $C_{\text{src}} = \texttt{while x do skip};C$ and $C_{\text{tgt}} = \texttt{if x = 0 then } C \texttt{ else while } \mathit{true} \texttt{ do skip}$, where C is an arbitrary loop-free command, we have:

- $C_{\text{src}} \rightsquigarrow C_{\text{tgt}}$, as $\lfloor C_{\text{tgt}} \rfloor = \{\langle s, \theta, e\rangle ; t \mid e(s)(\texttt{x}) = 0, t \in \lfloor C \rfloor\} \subseteq \lfloor C_{\text{src}} \rfloor$.
- If $\langle P[C_{\text{tgt}}], s\rangle \downarrow \infty$, then $\langle P[C_{\text{src}}], s\rangle \downarrow \infty$. To show this, if C_{tgt} is executed when $s(\texttt{x}) = 0$, then we execute the cycle for C_{src} and skip it, and we will have C on both sides. If C_{tgt} is executed when $s(\texttt{x}) \neq 0$, then for C_{src} we can enter the cycle for forever.
- $C_{\text{src}} \not\rightsquigarrow_{\text{fair}} C_{\text{tgt}}$, because for $P = - \parallel \texttt{x} := 1$ we have $\langle P[C_{\text{tgt}}], s_0\rangle \downarrow_{\text{fair}} \infty$ and $\langle P[C_{\text{src}}], s_0\rangle \not\downarrow_{\text{fair}} \infty$.

D FINITE ADEQUACY

Proposition D.1. *If* C_{tgt} *is loop-free and* $\lfloor C_{\text{tgt}} \rfloor \subseteq \lfloor C_{\text{src}} \rfloor$*, then* $C_{\text{src}} \rightsquigarrow_{\text{fairseq}} C_{\text{tgt}}$.

Proof. If C is loop-free, then $\lfloor C \rfloor_\infty$ is empty. So this follows from Thm. D.11. **O**: *can we prove this without* $\lfloor \cdot \rfloor_\infty$*? The claim has nothing to do with* $\lfloor \cdot \rfloor_\infty$. **M**: *we can use the next proof without rewriting traces, though that proof is still heavy* □

Proposition D.2. *If* C_{tgt} *is loop-free and* $\lfloor C_{\text{tgt}} \rfloor \subseteq [\![C_{\text{src}}]\!]$*, then* $C_{\text{src}} \rightsquigarrow_{\text{fairseq}} C_{\text{tgt}}$.

Proof. **O**: *this proof is too high level and will be impossible to mechanize... It has to be related to inductive/co-inductive definitions from before* Consider a fair infinite computation of $P[C_{\text{tgt}}]$. By Lemmas A.2, A.3 and A.5, we can split this computation into segments of two types:

- From $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s\rangle$ to $\langle \mathbb{P}'[C_{\text{tgt}}](\texttt{skip}) = \mathbb{P}'\{\texttt{skip}/\bowtie\}[C_{\text{tgt}}](C_{\text{tgt}}), s'\rangle$;
- From $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s\rangle$ to the end, in such a way that the second case of Lemma A.5 holds at every step.

Indeed, if we have $\mathbb{P}[C_{\mathrm{tgt}}](C')$ at some point, and $C' \neq \mathtt{skip}$ and $C' \neq C_{\mathrm{tgt}}$, the first case of Lemma A.5 should hold at some moment after that—otherwise fairness is violated; and there is a finite number of such C' as C_{tgt} is loop-free.

If $\langle \mathbb{P}[C_{\mathrm{tgt}}](C_{\mathrm{tgt}}), s\rangle \to^* \langle \mathbb{P}'[C_{\mathrm{tgt}}](\mathtt{skip}), s'\rangle$, then this segment is described[2] by some $t \in \lfloor C_{\mathrm{tgt}} \rfloor$. Then there exists a trace $t' \in \lfloor C_{\mathrm{src}} \rfloor$ such that $t' \rightrightarrows t$. We observe that t' has the same environment actions as t, and just before every action $\bar{\mathtt{W}}(x, v)$ we have the same value of x in both t and t'. It follows from Lemma A.1 that $\langle \mathbb{P}[C_{\mathrm{src}}](C_{\mathrm{src}}), s\rangle \to^* \langle \mathbb{P}'[C_{\mathrm{src}}](\mathtt{skip}), s'\rangle$.

If $\langle \mathbb{P}[C_{\mathrm{tgt}}](C_{\mathrm{tgt}}), s\rangle \to \langle \mathbb{P}'[C_{\mathrm{tgt}}](C_{\mathrm{tgt}}), s'\rangle$, then we have $\langle \mathbb{P}[C_{\mathrm{src}}](C_{\mathrm{src}}), s\rangle \to \langle \mathbb{P}'[C_{\mathrm{src}}](C_{\mathrm{src}}), s'\rangle$.

To show that these steps together form a fair infinite computation of $P[C_{\mathrm{src}}]$, it suffices to observe that, during one segment of the first type, C_{src} does a finite number of component actions. So we can have enough fuel for it to execute; the other threads will be the same (and will spend the same amount of fuel) as in fair infinite computation of $P[C_{\mathrm{tgt}}]$. Finally, this fair infinite computation visits the same sequence of states—it follows from Prop. 4.19 and from the observation that, if $t \xrightarrow{\mathcal{R}^{\mathrm{alt}}} t'$, then $|t'| \leq 1$.

□

M: *another proof for Prop. D.2* For every C—a subcommand of C_{tgt}, we define a relation $\rhd_C \triangleq \{\langle\langle \mathbb{P}[C_{\mathrm{tgt}}](C), s\rangle, \langle \mathbb{P}[C_{\mathrm{src}}](C'(C, s)), s\rangle\rangle \mid \mathbb{P} \in \mathrm{Gctx}, s \in \mathrm{State}\}$ (here C' is a subcommand of C_{src}, it will be decided later which one), and $\rhd$ is a union of all such relations. We also choose C' such that, if $t = \langle s, \theta, c\rangle$ and $t \in \lfloor C \rfloor$, then $t \in [\![C']\!]$.

Claim D.2.1: If $\langle C_1, s_1\rangle \to \langle C_1', s_1'\rangle$, and $\langle C_1, s_1\rangle \rhd_C \langle C_2, s_2\rangle$, then there exist a tuple $\langle C_2', s_2'\rangle$ and a command C' such that $\langle C_1', s_1'\rangle (\rhd_C \cup \rhd_{C'}) \langle C_2', s_2'\rangle$ and $\langle C_2, s_2\rangle \to^* \langle C_2', s_2'\rangle$.

Proof. We prove this proposition by induction on subcommands of C_{tgt}, in the execution order from C_{tgt} to $\mathtt{skip}$. We observe that $C_1 = \mathbb{P}[C_{\mathrm{tgt}}](C)$, $C_2 = \mathbb{P}[C_{\mathrm{src}}](C'')$ and $s_1 = s_2$. For the base case, we define $C'(C_{\mathrm{tgt}}, s) = C_{\mathrm{src}}$. Otherwise, by induction hypothesis, we can assume that C'' is a certain subcommand of C_{src}.

First, suppose that $C_1 \neq \mathbb{P}[C_{\mathrm{tgt}}](\mathtt{skip})$. We apply Lemma A.5 to the step $\langle \mathbb{P}[C_{\mathrm{tgt}}](C), s_1\rangle \xrightarrow{\gamma} \langle C_1', s_1'\rangle$. If $\gamma = \mathtt{W}(x, v)$ and the first case holds, then we define $\alpha = \mathtt{W}(x, v)$; if $\gamma = \mathtt{W}(x, v)$ and the second case holds, then we define $\alpha = \bar{\mathtt{W}}(x, v)$; otherwise $\alpha = \varepsilon$. The first case also provides some C' such that $\langle C, s_1\rangle \xrightarrow{\gamma} \langle C', s_1'\rangle$; if the second case holds, then we have no C'.

We take an arbitrary trace $t \in \lfloor C \rfloor$ that starts with α from the state s_1.

By induction hypothesis, we observe that $t \in [\![C'']\!]$, so there exists a trace $t' \in \lfloor C'' \rfloor$ such that $t' \rightrightarrows t$. By Prop. A.20, there exist traces t_1', t_2' such that $t' = t_1' ; t_2'$ and $t_1' \rightrightarrows \langle s_1, \theta, \alpha\rangle$. Let C_1'' be defined by $\langle C'', s_1\rangle \overset{t_1'}{\Longrightarrow} \langle C_1'', s_1'\rangle$ (such exists due to Claim 3.4.2 and that $\overset{c}{\Longrightarrow}$ is defined inductively). We observe that $\langle s_1', \theta, t_2'\rangle \in \lfloor C_1'' \rfloor$.

If the first case holds, we take $C_2' = \mathbb{P}[C_{\mathrm{src}}](C_1'')$ and $s_2' = s_1'$. If the second case holds, we take $C_2' = \mathbb{P}'[C_{\mathrm{src}}](C_1'')$ and $s_2' = s_1'$.

It remains to show that, if $t_0 = \langle s_1', \theta, c\rangle$ and $t_0 \in \lfloor C \rfloor$ (for the second case) or $t_0 \in \lfloor C' \rfloor$ (for the first case), then $t_0 \in [\![C_1'']\!]$. In either case, we have $\langle s_1, \theta, \alpha \cdot c\rangle \in \lfloor C \rfloor \subseteq [\![C'']\!]$, so by repeating the argument in the previous pagagraph we have $t_2' \rightrightarrows \langle s_1', \theta, c\rangle$ and therefore $t_0 \in [\![C_1'']\!]$. **M**: *this is wrong: maybe it is a different t_2', which is not in $\lfloor C_1'' \rfloor$ for C_1'' that we have chosen before*

[2] By Lemma A.5, we can establish which steps are made by C_{tgt} and which by context; we write down the latter as environment actions. Thus, we obtain a chronicle c such that $\langle C_{\mathrm{tgt}}, s\rangle \overset{c}{\Longrightarrow} \langle \mathtt{skip}, s'\rangle$; it follows from Claim 3.4.2 that for some store θ we have $\langle s, \theta, c\rangle \in \lfloor C_{\mathrm{tgt}} \rfloor$.

For the case when $C' = \texttt{skip}$, we can take $t = \langle s_1, \theta, \alpha \rangle$, and our argument will give $C_1'' = \texttt{skip}$. This way, if $C_1 = \mathbb{P}[C_{\text{tgt}}](\texttt{skip})$ and $\langle C_1, s_1 \rangle \rhd_C \langle C_2, s_2 \rangle$, then $C_2 = \mathbb{P}[C_{\text{src}}](\texttt{skip})$, and we are done as $\lfloor \texttt{skip} \rfloor \subseteq [\![\texttt{skip}]\!]$. Next, we can transform $\mathbb{P}[C_{\text{tgt}}](\texttt{skip})$ into $\mathbb{P}\{\texttt{skip}/\bowtie\}[C_{\text{tgt}}](C_{\text{tgt}})$, transforming $\mathbb{P}[C_{\text{src}}](\texttt{skip})$ at the same time—this works as $\lfloor C_{\text{tgt}} \rfloor \subseteq [\![C_{\text{src}}]\!]$. □

Claim D.2.2: If $\langle P[C_{\text{tgt}}], s \rangle \downarrow_{\text{fair}} s_1, s_2, \dots, s_n, \dots$, then $\langle P[C_{\text{src}}], s \rangle \downarrow_{\text{fair}} s_1, s_2, \dots, s_n, \dots$.

Proof. Since $\rhd$ is a simulation relation and $\langle P[C_{\text{tgt}}], s \rangle \rhd \langle P[C_{\text{src}}], s \rangle$, we have: if $\langle P[C_{\text{tgt}}], s \rangle \downarrow_{\text{fair}} s_1, s_2, \dots, s_n, \dots$, then there exists an infinite computation of $\langle P[C_{\text{src}}], s \rangle$ visiting the same states. To make sure that this computation is fair, we will use name trees.

We use some fresh names for threads inside C_{src} (or its subcommands) and the same names for other threads.

We recall that infinite run is a sequence of the form $\langle C, s, \mathbf{N} \rangle \xrightarrow{N_i} \langle C', s', \mathbf{N}' \rangle$, and it is fair if $\forall \nu \in \text{Name} \forall n \in \mathbb{N}. \exists k > n. \nu \notin N_k$. We write N_i for inactive names sets in the steps of $P[C_{\text{tgt}}]$ and N_i'—for $P[C_{\text{src}}]$.

Suppose ν is a name for some thread in $\mathbb{P}$. Then, as the computation of $P[C_{\text{tgt}}]$ is fair, we have $\forall n \in \mathbb{N}. \exists k > n. \nu \notin N_k$. So for every natural n, we can find a number k' such that $k' > n$, the step k in the computation of $P[C_{\text{tgt}}]$ corresponds to the step k' in the computation of $P[C_{\text{src}}]$ (maybe together with other steps), and $\nu \notin N_k$. As N_k is similar to $N_{k'}'$ (except for thread names of C_{tgt} and C_{src}), we have $\nu \notin N_{k'}'$.

Suppose ν is a name for some thread in C_{src}. If $\nu \in N_{n'}'$, then this n'th step of $P[C_{\text{src}}]$ corresponds to nth step of $P[C_{\text{tgt}}]$, and let ν' be the name of 'main' thread for C_{tgt} used in that step. We observe that $\nu' \notin N_k$ holds for infinitely many k bigger than n; this holds as C_{tgt} takes steps (it can take finite number of steps, as it is loop-free), or because C_{tgt} reaches $\texttt{skip}$ (and since that step, ν' becomes out of use). At the step in $P[C_{\text{src}}]$ corresponding to the latter case, C_{src} also reaches $\texttt{skip}$, and ν becomes out of use.

We have shown that $\forall n \in \mathbb{N}. \exists k > n. \nu \notin N_k'$ holds for all names in $P[C_{\text{src}}]$, so it is fair as required. □

Proposition D.3. *If C_{tgt} is loop-free and $\lfloor C_{\text{tgt}} \rfloor \subseteq [\![C_{\text{src}}]\!]$, then for every context P such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, $\langle P[C_{\text{tgt}}], s \rangle \downarrow s_1, s_2, \dots, s_n, \dots$ implies $\langle P[C_{\text{src}}], s \rangle \downarrow s_1, s_2, \dots, s_n, \dots$.*

Proof. The only difference from the previous proposition is that the infinite computation of $P[C_{\text{tgt}}]$ may be not fair. Therefore, four types of segments are possible:

- From $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s \rangle$ to $\langle \mathbb{P}'[C_{\text{tgt}}](\texttt{skip}) = \mathbb{P}'\{\texttt{skip}/\bowtie\}[C_{\text{tgt}}](C_{\text{tgt}}), s' \rangle$;
- From $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s \rangle$ to the end, in such a way that the second case of Lemma A.5 holds at every step;
- From $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s \rangle$ to $\langle \mathbb{P}'[C_{\text{tgt}}](C'), s' \rangle$;
- From $\langle \mathbb{P}[C_{\text{tgt}}](C'), s \rangle$ to the end, in such a way that the second case of Lemma A.5 holds at every step.

In the last two cases, C' is some subcommand of C_{tgt} different from C_{tgt} and $\texttt{skip}$. We also observe that the third-type segment can appear in the computation only once.

Our construction of an infinite computation of $P[C_{\text{src}}]$ is the same, though we need a new solution for the third case. For it, we have a trace $t = t_1 ; t_2$ such that t_1 describes the segment $\langle \mathbb{P}[C_{\text{tgt}}](C_{\text{tgt}}), s \rangle \rightarrow^* \langle \mathbb{P}'[C_{\text{tgt}}](C'), s' \rangle$, and $t \in \lfloor C_{\text{tgt}} \rfloor$. Then there exists a trace $t' \in \lfloor C_{\text{src}} \rfloor$ such that $t' \rightrightarrows t$; by Prop. A.20, there exist traces t_1', t_2' such that $t' = t_1' ; t_2'$ and $t_1' \rightrightarrows t_1$. It follows that t_1' describes the segment $\langle \mathbb{P}[C_{\text{src}}](C_{\text{src}}), s \rangle \rightarrow^* \langle \mathbb{P}'[C_{\text{src}}](C''), s' \rangle$, where C'' is some subcommand of C_{src}, and the sequence of visited states is also preserved. □

$$\frac{e \in \mathsf{EnvChro} \quad \langle \theta(L), e(s)\rangle \xrightarrow{\gamma} \langle v, s'\rangle \quad \langle s', \theta[a \mapsto v], c\rangle \in \lfloor C \rfloor_\infty}{\langle s, \theta, e \cdot \gamma \cdot c\rangle \in \lfloor \mathtt{let}\ a = L\ \mathtt{in}\ C \rfloor_\infty} \qquad \frac{t_1 \in \lfloor C_1 \rfloor_\infty}{t_1 \in \lfloor C_1 ; C_2 \rfloor_\infty} \qquad \frac{t_1 \in \lfloor C_1 \rfloor \quad t_2 \in \lfloor C_2 \rfloor_\infty}{t_1 ; t_2 \in \lfloor C_1 ; C_2 \rfloor_\infty}$$

$$\frac{\omega \in \mathsf{InfEnvChro} \quad t_1 \in \lfloor C_1 \rfloor_\infty \quad t_2 = \langle s, \theta, c_2\rangle \in \lfloor C_2 \rfloor}{t_1 \parallel (t_2 ; \langle c_2(s), \theta, \omega\rangle) \in \lfloor C_1 \parallel C_2 \rfloor_\infty} \qquad \frac{t_1 = \langle s, \theta, c_1\rangle \in \lfloor C_1 \rfloor \quad \omega \in \mathsf{InfEnvChro} \quad t_2 \in \lfloor C_2 \rfloor_\infty}{(t_1 ; \langle c_1(s), \theta, \omega\rangle) \parallel t_2 \in \lfloor C_1 \parallel C_2 \rfloor_\infty} \qquad \frac{t_1 \in \lfloor C_1 \rfloor_\infty \quad t_2 \in \lfloor C_2 \rfloor_\infty}{t_1 \parallel t_2 \in \lfloor C_1 \parallel C_2 \rfloor_\infty}$$

$$\frac{t \in \lfloor C_1 \rfloor_\infty \cup \lfloor C_2 \rfloor_\infty}{t \in \lfloor C_1 \oplus C_2 \rfloor_\infty} \qquad \frac{\theta(E) \neq 0 \implies \langle s, \theta, c\rangle \in \lfloor C_1 \rfloor_\infty \quad \theta(E) = 0 \implies \langle s, \theta, c\rangle \in \lfloor C_2 \rfloor_\infty}{\langle s, \theta, c\rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ C_1\ \mathtt{else}\ C_2 \rfloor_\infty} \qquad \frac{\langle e(s), \theta, c\rangle \in \lfloor C \rfloor \quad e \in \mathsf{EnvChro} \quad e(s)(x) \neq 0 \quad t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor_\infty}{\langle s, \theta, e \cdot c\rangle ; t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor_\infty}$$

$$\frac{\langle e(s), \theta, c\rangle \in \lfloor C \rfloor_\infty \quad e \in \mathsf{EnvChro} \quad e(s)(x) \neq 0}{\langle s, \theta, e \cdot c\rangle \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ C \rfloor_\infty} \qquad \frac{t_1 \in \lfloor C \rfloor_\infty}{t_1 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor_\infty} \qquad \frac{t_1 \in \lfloor C \rfloor \quad t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor_\infty}{t_1 ; t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ C \rfloor_\infty}$$

Fig. 9. Concrete Trace Semantics: $t \in \lfloor C \rfloor_\infty$

D.1 Concrete Denotational Semantics

By Fig. 9, we introduce the new concrete semantics $\lfloor C \rfloor_\infty$ which intuitively describes possible fair infinite executions of C.

We have the following differences from previous semantics:

- Chronicles may now be infinite. We write InfEnvChro for the set of all (possibly infinite) environment chronicles.
- The composition $t_1 \parallel t_2$ is defined only when t_1, t_2 are both infinite or have the same finite length.
- The set $\lfloor C \rfloor_\infty$ is defined coinductively; i.e., it is the greatest set satisfying the rules in Fig. 9.

Example D.4. We always have $C_1 \parallel C_2 \rightsquigarrow C_1 ; C_2$, but, when C_1 has loops, we may have $C_1 \parallel C_2 \not\rightsquigarrow_{\mathsf{fair}} C_1 ; C_2$.

Suppose $t_1 ; t_2 \in \lfloor C_1 ; C_2 \rfloor$ for some $t_1 \in \lfloor C_1 \rfloor, t_2 \in \lfloor C_2 \rfloor$. Let t_1' be the same trace as t_1, but with all environment actions, and similar for t_2'. Then we have $t_1 ; t_2' \in \lfloor C_1 \rfloor$ and $t_1' ; t_2 \in \lfloor C_2 \rfloor$, so $t_1 ; t_2 \in \lfloor C_1 \parallel C_2 \rfloor$.

This argument fails to show $\lfloor C_1 ; C_2 \rfloor_\infty \subseteq \lfloor C_1 \parallel C_2 \rfloor_\infty$ when $\lfloor C_1 \rfloor_\infty \neq \emptyset$.

Lemma D.5. *If $\lfloor C_1 \rfloor_\infty \subseteq \lfloor C_2 \rfloor_\infty$, then $\lfloor P[C_1] \rfloor_\infty \subseteq \lfloor P[C_2] \rfloor_\infty$ for every context P.*

Adequacy. To prove an infinite analogue of Lemma 3.4, we define coinductively a new relation $\overset{c}{\Longrightarrow}$, which describes interrupted infinite runs:

$$\frac{m < n \quad \alpha = \bar{\mathtt{W}}(x, v) \quad \langle C, s[x \mapsto v], T, m\rangle \overset{c}{\Longrightarrow} \infty}{\langle C, s, T, n\rangle \overset{\alpha \cdot c}{\Longrightarrow} \infty} \qquad \frac{m > n \quad \langle C, s, T\rangle \xrightarrow{\varepsilon} \langle C', s, T'\rangle \quad \langle C', s', T', m\rangle \overset{c}{\Longrightarrow} \infty}{\langle C, s, T, n\rangle \overset{c}{\Longrightarrow} \infty} \qquad \frac{m > n \quad \alpha = \mathtt{W}(x, v) \quad \langle C, s, T\rangle \xrightarrow{\alpha} \langle C', s', T'\rangle \quad \langle C', s', T', m\rangle \overset{c}{\Longrightarrow} \infty}{\langle C, s, T, n\rangle \overset{\alpha \cdot c}{\Longrightarrow} \infty}$$

Lemma D.6. *For a closed command C, we have $\langle C, s\rangle \downarrow_{\mathsf{fair}} \infty$ iff $\langle s, \theta, c\rangle \in \lfloor C \rfloor_\infty$ for some store θ and component chronicle c.*

Also, we have $\langle C, s\rangle \downarrow_{\mathsf{fair}} s_1, s_2, , \ldots, s_n, \ldots$ *iff* $\langle s, \theta, c\rangle \in \lfloor C \rfloor_\infty$ *for some store* θ *and component chronicle* c*, such that the sequence of prefixes of* c *applied to* s *contains the sequence of* s_i*.*

Claim D.6.1: $\langle C, s\rangle \downarrow_{\mathsf{fair}} \infty$ iff $\langle C, s, \langle n\rangle, m\rangle \overset{c}{\Rightarrow} \infty$ for some component chronicle $c \in \mathsf{CmpChro}$ and $n, m \in \mathbb{N}$.

Also, $\langle C, s\rangle \downarrow_{\mathsf{fair}} s_1, s_2, , \ldots, s_n, \ldots$ iff $\langle C, s, \langle n\rangle, m\rangle \overset{c}{\Rightarrow} \infty$ for some component chronicle $c \in \mathsf{CmpChro}$ and $n, m \in \mathbb{N}$, such that the sequence of prefixes of c applied to s contains the sequence of s_i.

Proof. Suppose $\langle C, s, \langle n\rangle, m\rangle \overset{c}{\Rightarrow} \infty$. To make such a conclusion from coinductive definition of $\overset{c}{\Rightarrow}$, we provide an infinite proof, which does not use the first rule as c is a component chronicle. For each step in this infinite proof, let us write down the corresponding step in fuel operational semantics in the form $\langle C, s, T\rangle \to \langle C', s', T'\rangle$. We would obtain an infinite execution in fuel operational semantics, thus $\langle C, s\rangle \downarrow_{\mathsf{fair}} \infty$. The other direction is proven in the similar way. □

Claim D.6.2: Let C be a command, and let $a_1, \ldots, a_k$ be an enumeration of $\mathrm{fv}(C)$. Then, $\langle s, \theta, c\rangle \in \lfloor C \rfloor_\infty$ iff $\langle C\{\theta(a_1)/a_1\} \ldots \{\theta(a_k)/a_k\}, s, \langle n\rangle, m\rangle \overset{c}{\Rightarrow} \infty$.

We prove this claim by coinduction on $\lfloor C \rfloor_\infty$. For example, when $C = C_1 \parallel C_2$, we need the following two lemmas:

Lemma D.7. *Suppose* $\langle C_1, s, T_1, m_1\rangle \overset{c_1}{\Longrightarrow} \infty$, $\langle C_2, s, T_2, m_2\rangle \overset{c_2}{\Longrightarrow} \infty$, *and* $c = c_1 \parallel c_2$*. Then for any natural numbers* n *and* $r_1 \geq m_2, r_2 \geq m_1$ *and* $m_0 \geq m_1, m_2$*, we have* $\langle C_1 \parallel C_2, \langle n, T_1[r_1], T_2[r_2]\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$.

Proof. We construct an infinite proof for $\langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$, using the infinite proofs for $\langle C_1, s, T_1, m_1\rangle \overset{c_1}{\Longrightarrow} \infty$ and $\langle C_2, s, T_2, m_2\rangle \overset{c_2}{\Longrightarrow} \infty$. We denote $c = \alpha \cdot c'$, where $\alpha = \mathtt{W}(x, v)$ or $\alpha = \bar{\mathtt{W}}(x, v)$; same for c_1', c_2'.

Suppose we have an environment step in both proofs (and thus the first action in both c_1, c_2 is $\bar{\mathtt{W}}(x, v)$). This means C_i (for $i = 1, 2$) matches the conclusion of the first rule with the clauses $\alpha = \bar{\mathtt{W}}(x, v), m_i' < m_i, \langle C_i, s[x \mapsto v], T_i, m_i'\rangle \overset{c_i'}{\Longrightarrow} \infty$. Let $m_0' = m_0 - 1$. Then $\langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$ matches the conclusion of the first rule with the clauses $\alpha = \bar{\mathtt{W}}(x, v), m_0' < m_0$ and $\langle C_1 \parallel C_2, s[x \mapsto v], \langle n, T_1[r_1], T_2[r_2]\rangle, m_0'\rangle \overset{c'}{\Longrightarrow} \infty$, as $r_1 \geq m_2', r_2 \geq m_1'$ and $m_0' \geq m_1', m_2'$.

Suppose we have an environment step for C_1 and a component step for C_2. Then we have $\langle C_2, s, T_2[r_2]\rangle \xrightarrow{\alpha} \langle C_2', s', T_2'[r_2]\rangle$ and $\langle C_2', s', T_2'[r_2], m_2'\rangle \overset{c_2'}{\Longrightarrow} \infty$ for some $m_2' > m_2$ and $\langle C_1, s', T_1[r_1], m_1'\rangle \overset{c_1'}{\Longrightarrow} \infty$ for some $m_1' < m_1$.

From fuel operational semantics we obtain $\langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle\rangle \xrightarrow{\alpha} \langle C_1 \parallel C_2', s', \langle n, T_1[r_1'], T_2'[r_2']\rangle\rangle$ for any $r_1' > r_1, r_2' < r_2$.

Let $r_1' = r_1 + (m_2' - m_2)$, $r_2' = r_2 + (m_1' - m_1)$ and $m_0' = m_0 + (m_2' - m_2)$. Then $\langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$ matches the conclusion of the third rule with the clauses $\alpha = \mathtt{W}(x, v), \langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle\rangle \xrightarrow{\alpha} \langle C_1 \parallel C_2', s', \langle n, T_1[r_1'], T_2'[r_2']\rangle\rangle, m_0' > m_0$ and $\langle C_1 \parallel C_2', s', \langle n, T_1[r_1'], T_2'[r_2']\rangle, m_0'\rangle \overset{c'}{\Longrightarrow} \infty$. We check that we can apply the lemma again for the last clause: $r_1' \geq m_2', r_2' \geq m_1'$ and $m_0' \geq m_1', m_2'$.

The case of component step for C_1 is symmetric.

Suppose we have an ε step for C_2 and no step for C_1. Then $\langle C_1 \parallel C_2, s, \langle n, T_1[r_1], T_2[r_2]\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$ matches the conclusion of the second rule, similar to previous case.

The case of ε step for C_1 is again symmetric. □

To conclude that also $\langle C_1 \parallel C_2, s, \langle n\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$ follows from $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c_1}{\Longrightarrow} \infty$, $\langle C_2, s, \langle n_2\rangle, m_2\rangle \overset{c_2}{\Longrightarrow} \infty$ and $c = c_1 \parallel c_2$, we can take $T_1 = \langle n_1\rangle, T_2 = \langle n_2\rangle$ and $n = max(n_1, n_2), m_0 = max(m_1, m_2)$. We do an extra step $\langle C_1 \parallel C_2, s, \langle n\rangle\rangle \overset{\varepsilon}{\to} \langle C_1 \parallel C_2, s, \langle n, \langle n\rangle, \langle n\rangle\rangle\rangle$ and then follow the lemma.

Lemma D.8. *Suppose* $\langle C_1 \parallel C_2, s, \langle n\rangle, m_0\rangle \overset{c}{\Rightarrow} \infty$. *Then we have either:*

- $c = c_1 \parallel c_2$, *and* $\langle C_i, s, \langle n_i\rangle, m_i\rangle \overset{c_i}{\Longrightarrow} \infty$ *for some natural numbers* n_i, m_i *and for* $i = 1, 2$;
- *or* $c = c_1 \parallel (c_2{\cdot}\omega)$ *for some* $\omega \in \mathsf{InfEnvChro}$, *and* $\langle C_2, s\rangle \overset{c_2}{\Longrightarrow} \langle \mathtt{skip}, c_2(s)\rangle$ *and* $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c_1}{\Longrightarrow} \infty$;
- *or symmetric to the second case.*

Proof. First we observe that $\langle C_1 \parallel C_2, s\rangle \downarrow_{\mathsf{fair}} \infty$, by Claim D.6.1, and the first step of this fair infinite execution is $\langle C_1 \parallel C_2, s, \langle n\rangle\rangle \overset{\varepsilon}{\to} \langle C_1 \parallel C_2, s, \langle n, \langle n\rangle, \langle n\rangle\rangle\rangle$. Suppose neither of these two threads dies out; then we can "extract" the steps of the form $\langle C_1, s, T_1\rangle \overset{\gamma}{\to} \langle C_1', s', T_1'\rangle$ from clauses for the steps $\langle C_1 \parallel C_2, s, \langle n_0, T_1, T_2\rangle\rangle \overset{\gamma}{\to} \langle C_1' \parallel C_2, s', \langle n_0, T_1'', T_2'\rangle\rangle$. Therefore we have a fair infinite execution for C_1, and in a similar way for C_2. Then $\langle C_i, s, \langle n_i\rangle, m_i\rangle \overset{c_i}{\Longrightarrow} \infty$ follows from Claim D.6.1. Suppose the right thread (of C_2) dies out; then we extract the steps of C_1 and of C_2 until that moment exactly as above, and all steps after that are the steps of C_1. So we have a finite execution of C_2 and an infinite fair execution of C_1; thus the second case takes place. The case when C_1 dies out is symmetric. □

We discuss also the case $C = C_1 \,;\, C_2$.

Lemma D.9. *Suppose* $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$. *Then we have* $\langle C_1 \,;\, C_2, s, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$.

Proof. We have an infinite proof for $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$; it suffices to rewrite each step of it applying the replacement $C \mapsto C \,;\, C_2$. □

Lemma D.10. *Suppose* $\langle C_1 \,;\, C_2, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$. *Then we have either:*

- $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$;
- *or* $c = c_1{\cdot}c_2$ *for some finite* c_1 *such that* $\langle C_1, s\rangle \overset{c_1}{\Longrightarrow} \langle \mathtt{skip}, c_1(s)\rangle$ *and* $\langle C_2, c_1(s), \langle n_2\rangle, m_2\rangle \overset{c_2}{\Longrightarrow} \infty$ *for some natural numbers* n_2, m_2.

Proof. First we observe that $\langle C_1 \,;\, C_2, s\rangle \downarrow_{\mathsf{fair}} \infty$, by Claim D.6.1. A step in this fair execution has the form $\langle C_1' \,;\, C_2, s', T\rangle \overset{\gamma}{\to} \langle C_1'' \,;\, C_2, s'', T'\rangle$, if and only if $\langle C_1', s, T\rangle \overset{\gamma}{\to} \langle C_1'', s'', T'\rangle$. If there are infinitely many such steps, then it provides a fair infinite execution of C_1, thus $\langle C_1, s, \langle n_1\rangle, m_1\rangle \overset{c}{\Rightarrow} \infty$ follows by Claim D.6.1. Otherwise, we observe that we cannot take steps of C_2 before C_1 will be all executed towards `skip`, so the second case takes place. □

Adequacy of the concrete semantics is now an immediate corollary:

Theorem D.11. *If* $\lfloor C_{\text{tgt}} \rfloor \subseteq \lfloor C_{\text{src}} \rfloor$ *and* $\lfloor C_{\text{tgt}} \rfloor_\infty \subseteq \lfloor C_{\text{src}} \rfloor_\infty$, *then* $C_{\text{src}} \rightsquigarrow_{\text{fairseq}} C_{\text{tgt}}$.

Proof. Suppose that $\lfloor C_{\text{tgt}} \rfloor \subseteq \lfloor C_{\text{src}} \rfloor$ and $\lfloor C_{\text{tgt}} \rfloor_\infty \subseteq \lfloor C_{\text{src}} \rfloor_\infty$. Let P be a context such that $P[C_{\text{src}}]$ and $P[C_{\text{tgt}}]$ are closed, and suppose that $\langle P[C_{\text{tgt}}], s\rangle \downarrow s'$. Since $\langle P[C_{\text{tgt}}], s\rangle \downarrow s'$, by Lemma 3.4, we have $\langle s, \theta, c\rangle \in \lfloor P[C_{\text{tgt}}] \rfloor$ for some store θ and finite component chronicle $c \in \mathsf{CmpChro}$ such that $c(s) = s'$. Since $\lfloor C_{\text{tgt}} \rfloor \subseteq \lfloor C_{\text{src}} \rfloor$, by Lemma D.5, it follows that $\langle s, \theta, c\rangle \in \lfloor P[C_{\text{src}}] \rfloor$. By Lemma 3.4, it follows that $\langle P[C_{\text{src}}], s\rangle \downarrow s'$.

Next, suppose that $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, , \ldots, s_n, \ldots$. By Lemma D.6, we have $\langle s, \theta, c\rangle \in \lfloor P[C_{\text{tgt}}] \rfloor_\infty$ for some store θ and component chronicle c, such that the sequence of prefixes of c applied to s contains the sequence of s_i. Since $\lfloor C_{\text{tgt}} \rfloor_\infty \subseteq \lfloor C_{\text{src}} \rfloor_\infty$, by Lemma D.5, it follows that $\langle s, \theta, c\rangle \in \lfloor P[C_{\text{src}}] \rfloor_\infty$. By Lemma D.6, it follows that $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} s_1, s_2, , \ldots, s_n, \ldots$.

TODO: *deal with "print" instructions* □

Example D.12. For the commands $C_{\text{src}} = \mathtt{z := 0} \parallel \mathtt{while\ z\ do}\ C_1 \parallel \mathtt{while} * \mathtt{do\ (if\ z = 0\ then}\ C_2\ \mathtt{else\ skip)}$ and $C_{\text{tgt}} = \mathtt{while} * \mathtt{do\ (assume(z} \neq 0)\ ; C_1)$, for some loop-free commands C_1, C_2, we have $\lfloor C_{\text{tgt}} \rfloor_\infty$ consisting of infinitely many repeats of $\lfloor C_1 \rfloor$, whether $\lfloor C_{\text{src}} \rfloor_\infty$ consists of finitely many repeats of $\lfloor C_1 \rfloor$ (though any finite number of repeats is possible), followed by $\mathtt{W(z, 0)}$ and then infinitely many repeats of $\lfloor C_2 \rfloor$.

Consider $C_1 = \mathtt{x := 1}\ ; \mathtt{assume(x = 0)}$, $C_2 = \mathtt{x := 2}$ and $C_3 = \mathtt{x := 0}\ ; \mathtt{assume(x = 1)}$. For $P = - \parallel \mathtt{while} * \mathtt{do}\ C_3$ and $s = \mathtt{s_0[z} \mapsto 1]$ we have $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} \infty$ and $\langle P[C_{\text{src}}], s\rangle \not\downarrow_{\text{fair}} \infty$.

On the other hand, for any loop-free context P', $\langle P'[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} \infty$ implies $\langle P'[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} \infty$.

So when we study the condition $\langle P[C_{\text{tgt}}], s\rangle \downarrow_{\text{fair}} \infty$ implies $\langle P[C_{\text{src}}], s\rangle \downarrow_{\text{fair}} \infty$, without specifying which states the corresponding infinite execution visits, we still need to examine at least $\lfloor C_{\text{tgt}} \rfloor_\infty \subseteq \lfloor C_{\text{src}} \rfloor_\infty$.

Proposition D.13. *For every command* C, *the set* $\{C' \mid \exists s, s'.\ \langle C, s\rangle \rightarrow^* \langle C', s'\rangle\}$ *is finite.*

Proposition D.14. *The following conditions on* C_1, C_2 *are equivalent:*

- *for every context* P, $\langle P[C_1], s\rangle \downarrow_{\text{fair}} \infty$ *implies* $\langle P[C_2], s\rangle \downarrow_{\text{fair}} \infty$;
- *for every context* P, $\langle P[C_1], s\rangle \downarrow_{\text{fair}} s_1, s_2, , \ldots, s_n, \ldots$ *implies* $\langle P[C_2], s\rangle \downarrow s_1, s_2, , \ldots, s_n, \ldots$ *and* $\langle P[C_2], s\rangle \downarrow_{\text{fair}} s_1', s_2', , \ldots, s_n', \ldots$.

Proof. The implication from the second to the first is trivial.

For the converse, suppose that $P, \langle P[C_1], s\rangle \downarrow_{\text{fair}} s_1, s_2, , \ldots, s_n, \ldots$. Then $\langle P[C_2], s\rangle \downarrow_{\text{fair}} s_1', s_2', , \ldots, s_n', \ldots$ follows. Next, for every natural number n we construct a context $P_n = P \parallel \mathtt{snapshot}(s_1)\ ; \mathtt{snapshot}(s_2); \ldots; \mathtt{snapshot}(s_n)$. We have $\langle P_n[C_1], s\rangle \downarrow_{\text{fair}} \infty$ and so $\langle P_n[C_2], s\rangle \downarrow_{\text{fair}} \infty$. From the fair infinite execution of $P_n[C_2]$, we can "extract" the finite execution of $P[C_2]$, which visits all the states from s_1 to s_n. By Prop. D.13, we can form an infinite subsequence of these finite executions, such that each next execution continues the previous one. The union of this subsequence is an infinite execution (not necessarily fair), which visits all the states s_i, so we conclude $\langle P[C_2], s\rangle \downarrow s_1, s_2, , \ldots, s_n, \ldots$. □

E CONTEXT SEMANTICS—INFINITE

In Fig. 10 we define $\lfloor P \rfloor_\infty$ with two arguments.

We say that $P_{\text{src}} \rightsquigarrow_{\text{fair}} P_{\text{tgt}}$ ($P_{\text{src}} \rightsquigarrow_{\text{fairseq}} P_{\text{tgt}}$), if for every command C we have $P_{\text{src}}[C] \rightsquigarrow_{\text{fair}} P_{\text{tgt}}[C]$ ($P_{\text{src}}[C] \rightsquigarrow_{\text{fairseq}} P_{\text{tgt}}[C]$).

Proposition E.1. *For every context* P *and command* C, *we have* $\lfloor P \rfloor_\infty(\lfloor C \rfloor, \lfloor C \rfloor_\infty) = \lfloor P[C] \rfloor_\infty$.

Proposition E.2. *If* $\lfloor P_{\text{tgt}} \rfloor_\infty(\lfloor C \rfloor, \lfloor C \rfloor_\infty) \subseteq \lfloor P_{\text{src}} \rfloor_\infty(\lfloor C \rfloor, \lfloor C \rfloor_\infty)$ *and* $\lfloor P_{\text{tgt}} \rfloor(\lfloor C \rfloor) \subseteq \lfloor P_{\text{src}} \rfloor(\lfloor C \rfloor)$ *for every command* C, *then* $P_{\text{src}} \rightsquigarrow_{\text{fairseq}} P_{\text{tgt}}$.

$$\frac{t \in T_2}{t \in \lfloor - \rfloor_\infty(T_1,T_2)} \qquad \frac{e \in \mathsf{EnvChro} \quad \langle \theta(L), e(s)\rangle \xrightarrow{\gamma} \langle v, s'\rangle \quad \langle s', \theta[a \mapsto v], c\rangle \in \lfloor P\rfloor_\infty(T_1,T_2)}{\langle s, \theta, e\cdot\gamma\cdot c\rangle \in \lfloor \mathtt{let}\ a = L\ \mathtt{in}\ P\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 \in \lfloor P\rfloor_\infty(T_1,T_2)}{t_1 \in \lfloor P\,;C\rfloor_\infty(T_1,T_2)}$$

$$\frac{t_1 \in \lfloor P\rfloor(T_1) \quad t_2 \in \lfloor C\rfloor_\infty}{t_1\,;t_2 \in \lfloor P\,;C\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 \in \lfloor C\rfloor_\infty}{t_1 \in \lfloor C\,;P\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 \in \lfloor C\rfloor \quad t_2 \in \lfloor P\rfloor_\infty(T_1,T_2)}{t_1\,;t_2 \in \lfloor C\,;P\rfloor_\infty(T_1,T_2)}$$

$$\frac{\omega \in \mathsf{InfEnvChro} \quad t_1 \in \lfloor C\rfloor_\infty \quad t_2 = \langle s,\theta,c_2\rangle \in \lfloor P\rfloor(T_1)}{t_1 \parallel (t_2\,;\langle c_2(s),\theta,\omega\rangle) \in \lfloor C \parallel P\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 = \langle s,\theta,c_1\rangle \in \lfloor C\rfloor \quad \omega \in \mathsf{InfEnvChro} \quad t_2 \in \lfloor P\rfloor_\infty(T_1,T_2)}{(t_1\,;\langle c_1(s),\theta,\omega\rangle) \parallel t_2 \in \lfloor C \parallel P\rfloor_\infty(T_1,T_2)}$$

$$\frac{t_1 \in \lfloor C\rfloor_\infty \quad t_2 \in \lfloor P\rfloor_\infty(T_1,T_2)}{t_1 \parallel t_2 \in \lfloor C \parallel P\rfloor_\infty(T_1,T_2)} \qquad \frac{}{\lfloor P \parallel C\rfloor_\infty = \lfloor C \parallel P\rfloor_\infty} \qquad \frac{t \in \lfloor C\rfloor_\infty \cup \lfloor P\rfloor_\infty(T_1,T_2)}{t \in \lfloor C \oplus P\rfloor_\infty(T_1,T_2)}$$

$$\frac{t \in \lfloor C\rfloor_\infty \cup \lfloor P\rfloor_\infty(T_1,T_2)}{t \in \lfloor P \oplus C\rfloor_\infty(T_1,T_2)} \qquad \frac{\theta(E) \neq 0 \implies \langle s,\theta,c\rangle \in \lfloor C\rfloor_\infty \quad \theta(E) = 0 \implies \langle s,\theta,c\rangle \in \lfloor P\rfloor_\infty(T_1,T_2)}{\langle s,\theta,c\rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ C\ \mathtt{else}\ P\rfloor_\infty(T_1,T_2)}$$

$$\frac{\theta(E) = 0 \implies \langle s,\theta,c\rangle \in \lfloor C\rfloor_\infty \quad \theta(E) \neq 0 \implies \langle s,\theta,c\rangle \in \lfloor P\rfloor_\infty(T_1,T_2)}{\langle s,\theta,c\rangle \in \lfloor \mathtt{if}\ E\ \mathtt{then}\ P\ \mathtt{else}\ C\rfloor_\infty(T_1,T_2)} \qquad \frac{\langle e(s),\theta,c\rangle \in \lfloor P\rfloor(T_1) \quad e \in \mathsf{EnvChro} \quad e(s)(x) \neq 0 \quad t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)}{\langle s,\theta,e\cdot c\rangle\,;t \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)}$$

$$\frac{\langle e(s),\theta,c\rangle \in \lfloor P\rfloor_\infty(T_1,T_2) \quad e \in \mathsf{EnvChro} \quad e(s)(x) \neq 0}{\langle s,\theta,e\cdot c\rangle \in \lfloor \mathtt{while}\ x\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 \in \lfloor P\rfloor_\infty(T_1,T_2)}{t_1 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)} \qquad \frac{t_1 \in \lfloor P\rfloor(T_1) \quad t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)}{t_1\,;t_2 \in \lfloor \mathtt{while}\ *\ \mathtt{do}\ P\rfloor_\infty(T_1,T_2)}$$

Fig. 10. Concrete Context Semantics: $t \in \lfloor P\rfloor_\infty(T_1,T_2)$

F ABSTRACT SEMANTICS

We adapt the definition from Lemma 4.26 for infinite traces.

A relation $\rightrightarrows_\infty$ between (possibly infinite) traces is coinductively defined as follows:

$$\frac{t_1 \xrightarrow{\mathcal{R}^{\mathsf{alt}}}{}^{?}\ t_1' \qquad t_2 \rightrightarrows_\infty t_2' \quad |t_1| + |t_1'| \geq 1}{t_1\,;t_2 \rightrightarrows_\infty t_1'\,;t_2'} \qquad \frac{}{\langle s,\theta,\varepsilon\rangle \rightrightarrows_\infty \langle s,\theta,\varepsilon\rangle}$$

where $\mathcal{R}^{\mathsf{alt}} \triangleq \{\xrightarrow{\mathsf{r}_1^{\mathsf{alt}}}, \xrightarrow{\mathsf{r}_2^{\mathsf{alt}}}, \xrightarrow{\mathsf{r}_3^{\mathsf{alt}}}, \xrightarrow{\mathsf{r}_4^{\mathsf{alt}}}\}$.

M: *a problem arises: by applying infinitely many* r_4*, I can rewrite* $\langle s,\theta,c\rangle$ *into* $\langle s,\theta,\mathtt{W}(\mathtt{x},s(\mathtt{x}))\cdot\mathtt{W}(\mathtt{x},s(\mathtt{x}))\cdots\rangle$ *for every* c*. By applying infinitely many* r_3*, I can rewrite in the opposite direction. So* $\rightrightarrows_\infty\,;\rightrightarrows_\infty$ *connects every pair of chronicles.*

A proof of $t_1 \rightrightarrows_\infty t_2$ is similar to an execution of two threads $\{1,2\}$. We say that the thread 1 *dies out* at some step, if the remainder of t_1 is empty, and that it is *taking step* if t_1' is not empty in $t_1' \xrightarrow{\mathcal{R}^{\mathsf{alt}}}{}^{?}\ t_2'$; and in the similar way for thread 2. We require the infinite proofs to be *fair*, in the sense that all alive threads should take infinitely many steps.

Proposition F.1. $\rightrightarrows_\infty\,;\rightrightarrows_\infty \subseteq \rightrightarrows_\infty$.

Proof. Suppose $t \rightrightarrows_\infty t' \rightrightarrows_\infty t''$, and in the proof of $t' \rightrightarrows_\infty t''$ the first step contains $t'_0 \xrightarrow{\mathcal{R}^{\text{alt}}}{}^{?} t''_0$. We construct a finite prefix $t_0 \sqsubseteq t$ such that $t_0 \rightrightarrows t''_0$.

To do this, we consider the steps in proof of $t \rightrightarrows_\infty t'$, which contain rewrites of the form $t_i \xrightarrow{\mathcal{R}^{\text{alt}}}{}^{?} t'_i$, and put them in one sequence until $|t'_1 ; t'_2; \dots ; t'_n| = |t'_0|$. If we cannot take the next step, as we have $|t'_1 ; t'_2; \dots ; t'_n| < |t'_0|$ and $|t'_1 ; t'_2; \dots ; t'_{n+1}| > |t'_0|$, we observe that this can only happen if $|t'_{n+1}| > 1$. This implies no rule from $\mathcal{R}^{\text{alt}}$ was applied, and $t_{n+1} = t'_{n+1}$. So we can split t_{n+1} into two parts to satisfy the length condition.

We have some finite prefix $t_0 \sqsubseteq t$ such that $t_0 \rightrightarrows t'_0 \rightrightarrows t''_0$. As for finite traces $\rightrightarrows$ is transitive (which follows from Lemma 4.26), we have $t_0 \rightrightarrows t''_0$; next we take steps of its proof and continue with some suffixes of t, t'', thus providing an infinite proof.

If $|t''_0| = 0$, this argument may result in $|t_0| = 0$, which is not sufficient. However, in this case t'_0 is not empty, and we have $t \rightrightarrows_\infty t'_1 \rightrightarrows_\infty t''$, where $t' = t'_0 ; t'_1$. As the proof for $t' \rightrightarrows_\infty t''$ is fair, this situation can hold only for a finite number of steps, and after that we can continue. □

We define $[\![C]\!]_\infty = \{t' \mid \exists t \in \lfloor C \rfloor_\infty . t \rightrightarrows_\infty t'\}$.