Concurrency in a Natural Semantics

Kevin Mitchell
LFCS, Department of Computer Science
University of Edinburgh

Abstract

Natural Semantics, a form of operational semantics, can be used to give a concise description of the evaluation of many sequential programming languages. However, concurrent languages present a problem for this approach as it is difficult to express the interaction between parallel processes. For such languages an approach such as Plotkin's structural operational semantics is usually preferred. Unfortunately such semantics are frequently verbose and may contain an unnecessary amount of detail about the evaluation. In this paper we show how the relational approach at the heart of Natural Semantics can be extended to handle concurrent languages and explore the relationship between these two semantic approaches in such a setting.

1 Introduction

A glance at the literature inspired by Plotkin's 1981 paper on structural operational semantics (SOS, [6]) reveals two distinct approaches to the choice of evaluation relation. In the "small-step" approach we construct a transition relation that corresponds to a single step in the computation process. The evaluation of the whole program then consists of a sequence of such steps. This approach has proved very successful, particularly when applied to process calculi such as CCS [4]. However, when the technique is extended to more complex sequential programming languages a number of deficiencies become apparent. For such languages we are often unconcerned about the intermediate steps; it is the final result of the evaluation that is of interest to us. The small step, or transitional, approach requires a large number of rules to propagate each small change of a sub-phrase into a corresponding change of the whole program. We can hide these propagation rules to some extent by the use of evaluation contexts [11], or a congruence relation [5]. However, this hiding is only partially successful as we still need to deal with the contexts or the congruence relation when reasoning about the semantics. A further problem can arise when some of the intermediate states in the computation do not correspond directly to programs in the language. In such cases we may need to extend the language with extra constructs before defining the semantics (e.g. [1]).

If we are only interested in the final state of a computation then we can simplify the semantics by defining a relation between programs and their computed values

$$\frac{e_1 \longrightarrow e_1'}{e_1 + e_2 \longrightarrow e_1' + e_2} \qquad \frac{e_1 \Longrightarrow \underline{n_1} \quad e_2 \Longrightarrow \underline{n_2}}{e_1 + e_2 \Longrightarrow \underline{n_1 + n_2}}$$

$$\frac{e_2 \longrightarrow e_2'}{e_1 + e_2 \longrightarrow e_1 + e_2'} \qquad \frac{}{\underline{n} \Longrightarrow \underline{n}}$$

$$\frac{}{\underline{n_1} + \underline{n_2} \longrightarrow \underline{n_1 + n_2}}$$

Small-step (transitional) Big-step (relational)

Figure 1: A choice of step size for the transition relation.

directly. This "big-step" style of semantics has also enjoyed some success (e.g. Natural Semantics [3]), particularly through its use in the Standard ML semantics [7]. These two styles of operational semantics are contrasted in Figure 1, where we define the evaluation of expressions consisting of either a natural number $\underline{n}$, or the sum of two expressions.

Concurrency is handled in a transitional semantics by the interleaving of the atomic actions of the processes. A relational semantics hides the intermediate steps in a computation within the proof tree, and so concurrent features can be difficult to handle in this approach. The drawback becomes more noticeable when attempting to extend sequential languages, such as Standard ML, that already have a semantics written in the relational style. At first glance it appears that we must recast the semantics in the transitional form before we can deal with any concurrent extensions [8]. Of course techniques exist for automatically performing such transformations in many cases [1]. However, the transformed semantics is often rather messy in comparison to the original, and this makes reasoning about the extended language more tedious. At the very least we would like to use the original semantics for reasoning about the sequential parts of the language, and the transformed semantics for the concurrent parts.

In this paper we investigate the extent to which the relational approach can handle concurrent languages. We start by introducing a small concurrent language supporting dynamic reconfiguration, and its transitional semantics. The language is based on the call-by-value lambda calculus, extended with facilities for spawning processes and communicating via message passing. We are interested in the problem of adding concurrency to existing, sequential, languages. The view taken is that parallelism is added in such cases either to exploit parallel hardware or to add expressive power to the language. Often the main aim of program evaluation remains unchanged — the calculation of a result. This view contrasts with that usually taken by proponents of process calculi, for example, where the interactions performed during the computation are of interest, and where the final result is frequently irrelevant. Our view influences the choice of concurrent operators, particularly the mechanism for creating processes. Inspired by CCS [4], we might introduce a new

construct $e_1 \mid e_2$ whose intended meaning is that the expressions e_1 and e_2 should be evaluated in parallel. What is the result of such an evaluation? Clearly there are a number of choices. We could wait for both e_1 and e_2 to terminate and then return the pair of values as result. However, it is often useful to be able to create a process that can outlive its creator. An example of this would be the creation of a register process that could be used instead of a reference in a language such as ML. We therefore choose an asymmetric version of parallel composition, written $e_1 \mid\rangle e_2$, meaning, as before, that e_1 and e_2 are evaluated in parallel. However, in this case the result of the evaluation will be the value returned by e_1. Furthermore, this value can be returned to the surrounding context *before* e_2 terminates, allowing e_2 to interact with the context later in the computation. This view is equivalent to introducing a spawn function for processes, as in Facile for example [8]. The asymmetry introduces the concept of a main thread of execution leading to the production of the final value. The spawned processes may influence the result by interacting with the main thread via message passing.

We demonstrate that it is possible to give a relational style of semantics for such a language and this forms the main contribution of the paper. The transitional and relational semantics can each be used to define the values computed by a program and we prove that these definitions agree with each other. These results are developed in two stages. We start by restricting our attention to those programs that completely terminate. We then relax this constraint slightly, allowing the spawned processes to diverge as long as the main thread terminates. Finally, we look at divergence and deadlock and explain why these properties can be difficult to deal with in the relational framework.

2 The Language

We assume an infinite set of ports, Ports, ranged over by p, and an infinite set of variables Vars, ranged over by x. The language is then defined by the following grammar:

Expressions	e	$::= a \mid e_1\, e_2 \mid e_1 \mid\rangle e_2 \mid i$
Atomics	a	$::= x \mid p \mid \lambda x.e$
Interactions	i	$::= a \Rightarrow e \mid a\, a' \Leftarrow e \mid i_1 + i_2$

The expression[1] $p \Rightarrow e$ is evaluated by first waiting for a value v to be received on port p. On arrival we then proceed to evaluate the expression $e\, v$, i.e. e should evaluate to an abstraction. The expression $p\, v \Leftarrow e$ transmits the value v on port p and then proceeds to evaluate e. The intention is that communications are synchronous — each input must match with a corresponding output, just as in the value passing variants of CCS [4]. A choice of communications is also possible, as

[1] We shall often blur the distinction between expressions, atomic expressions and interactions to simplify the presentation when this causes no confusion.

in $(p_1 \Rightarrow e_1) + (p_2\, v_2 \Leftarrow e_2)$. In this case at most one of the communications can succeed, the choice (if any) being made non-deterministically. Our language has no facility for creating local ports. Although they are extremely useful for pragmatic reasons they also complicate the analysis. The problems introduced by the scoping of channels seem to be largely orthogonal to the issues addressed in this report, and so they have been omitted for the sake of increased clarity.

Lambda abstraction is the only binding operator in the language, and so we shall use the terms *free*, *bound* and *closed* with their usual meanings. If we evaluate a closed expression e, and the computation terminates, then the resulting value will be either a port or a closed abstraction. We define the set of values by

Definition 2.1 $\quad v \in \mathsf{Values} ::= \mathsf{Ports} \cup \{\lambda x.\, e \mid \lambda x.\, e \text{ is closed}\}$

We will also assume the existence of a substitution operator, where $e\{v/x\}$ denotes the expression resulting from the replacement of all free occurrences of x in e by v. This is a rather simple form of substitution as v is always closed, i.e. no renaming is required.

When an expression reaches a state where an input or output is required then one can imagine the process issuing a request for communication. Such messages come in two flavours, depending on whether the expression wants to transmit or receive a value. We will use a set of *messages* to model these requests:

Definition 2.2 $\quad m \in \mathsf{Messages} ::= \underset{\substack{\uparrow \\ \textit{input}}}{p > v} \mid \underset{\substack{\uparrow \\ \textit{output}}}{p < v}$

We define a complement operator $\overline{}$ on messages as follows:

Definition 2.3 $\quad \overline{m} = \begin{cases} p < v, & \text{if } m = p > v; \\ p > v, & \text{if } m = p < v. \end{cases}$

Finally, we define the set of *actions*

Definition 2.4 $\quad \alpha \in \mathsf{Actions} ::= \mathsf{Messages} \cup \{\tau\}$

3 A Transitional Semantics

In this section we develop a small-step, transitional style of semantics for the language. We start by defining an evaluation context, $\mathcal{E}$ [11]. This will be used to control the order of subphrase evaluation.

$$\mathcal{E} ::= [\,] \mid \mathcal{E}\, e \mid (\lambda x.\, e)\, \mathcal{E} \mid \mathcal{E} \mid\rangle e \mid e \mid\rangle \mathcal{E}$$

An evaluation context is an expression with a hole in it (denoted by $[\,]$). We write $\mathcal{E}[e]$ for the expression resulting from filling in the hole with e.

input: $\dfrac{}{p \Rightarrow e \xrightarrow{p>v} e\,v}$ output: $\dfrac{}{p\,v \Leftarrow e \xrightarrow{p<v} e}$

synchronisation: $\dfrac{e_1 \xrightarrow{m} e_1' \quad e_2 \xrightarrow{\overline{m}} e_2'}{e_1 \,|\rangle\, e_2 \longrightarrow e_1' \,|\rangle\, e_2'}$

choice: $\dfrac{i_1 \xrightarrow{m} e}{i_1 + i_2 \xrightarrow{m} e}$ $\dfrac{i_2 \xrightarrow{m} e}{i_1 + i_2 \xrightarrow{m} e}$

application: $\dfrac{}{(\lambda x.e)\,v \longrightarrow e\{v/x\}}$

distribution:

$\dfrac{}{(\lambda x.e)\,(e_1 \,|\rangle\, e_2) \longrightarrow ((\lambda x.e)\,e_1) \,|\rangle\, e_2}$ $\dfrac{}{(e_1 \,|\rangle\, e_2)\,e_3 \longrightarrow (e_1\,e_3) \,|\rangle\, e_2}$

propagation: $\dfrac{e \xrightarrow{\alpha} e'}{\mathcal{E}[e] \xrightarrow{\alpha} \mathcal{E}[e']}$

Figure 2: The transitional semantics.

The semantics is given by a transition relation

$$\rightarrow \subseteq \text{Expressions} \times \text{Actions} \times \text{Expressions}$$

inductively defined by the set of rules in Figure 2, where we write $e \xrightarrow{\alpha} e'$ instead of $(e, \alpha, e') \in \rightarrow$, and use $e \longrightarrow e'$ instead of $e \xrightarrow{\tau} e'$. An example proof using this system is shown in Figure 3. Our semantics is only one of many we could give for this language. For example, we could allow e_1 and e_2 to be evaluated in parallel in the application $e_1\,e_2$. Going one step further, we could allow these parallel evaluations to communicate with each other. Such changes would introduce another way of creating processes and so we reject these variants on the grounds of simplicity. The technical results of this paper are not heavily dependent on the particular language, and so similar results could be developed for such variants.

$$\dfrac{\dfrac{\dfrac{}{p\,q \Leftarrow e_1 \xrightarrow{p<q} e_1}\,\text{output}}{(p\,q \Leftarrow e_1)\,r \xrightarrow{p<q} e_1\,r}\,\text{propagation} \quad \dfrac{}{p \Rightarrow e_2 \xrightarrow{p>q} e_2\,q}\,\text{input}}{((p\,q \Leftarrow e_1)\,r) \,|\rangle\, (p \Rightarrow e_2) \longrightarrow (e_1\,r) \,|\rangle\, (e_2\,q)}\,\text{synchronisation}$$

Figure 3: An example proof.

The overall effect of a computation is to reduce a program (i.e. a closed term) to a canonical form in which some quantity of information has been made explicit. Expression evaluation may result in subexpressions being evaluated in parallel. The process creation operator is asymmetric and so we may think of there being

a main thread of computation with subthreads being spun off as required. Under such a scheme we can either decide that a program terminates when the main thread reaches a canonical form, or we can demand that the whole expression must eventually reduce to a canonical form. We take the first interpretation, as we believe it to be more flexible, and is consistent with the view that it is the result of the main thread that is important. This motivates the following two definitions:

Definition 3.1 $\mathsf{main}(e) = \begin{cases} \mathsf{main}(e_1), & \text{if } e = e_1 \mid\rangle\, e_2; \\ e, & \text{otherwise.} \end{cases}$

Then

Definition 3.2 $\mathsf{eval}(e) = \{\nu : \mathsf{Values} \mid \exists e' \cdot e \longrightarrow^* e' \wedge \mathsf{main}(e') = \nu\}$

We will denote the set of values that can be produced using the second interpretation of evaluation by $\mathsf{eval}{\downarrow}\ (e)$.

Definition 3.3 $\mathsf{terminated}(e) = \begin{cases} \mathsf{terminated}(e_1) \wedge \mathsf{terminated}(e_2), & \text{if } e = e_1 \mid\rangle\, e_2; \\ e \in \mathsf{Values}, & \text{otherwise.} \end{cases}$

Definition 3.4

$$\mathsf{eval}\downarrow(e) = \{\nu : \mathsf{Values} \mid \exists e' \cdot e \longrightarrow^* e' \wedge \mathsf{main}(e') = \nu \wedge \mathsf{terminated}(e')\}$$

We could obviously develop a theory of the language at this point. However, we hope that the reader will agree that it captures the essence of at least part of more interesting concurrent languages, and will concentrate instead on developing an equivalent relational semantics.

4 A Relational Semantics

Consider for a moment the sequential subset of our language. A relational semantics for this fragment would usually be presented in the form of a relation $E \vdash e \Longrightarrow \nu$. This captures the notion of an expression e being evaluated in an environment E and producing a final value ν. Environments are just finite maps from variables to values, and determine the value of any free variables in e. In the full language this treatment is no longer acceptable. An expression may require some interaction with its surrounding context before it can yield a value. This naturally leads us to define relations of the form $E \vdash e \overset{t}{\Longrightarrow} \nu$, where t is a *trace* recording the external interactions required to produce the value ν. What are these traces? The following definition works well for our purposes:

Definition 4.1 $t \in \mathsf{Traces} ::= \varepsilon \mid m \bullet t \mid t_1 \mid\rangle\, t_2$

The sequential evaluation of two expressions naturally leads to the sequential composition of traces. This can be defined by cases as follows:

constant: $\dfrac{}{v \Longrightarrow v}$

input: $\dfrac{e\,v' \overset{t}{\Longrightarrow} v}{p \Rightarrow e \overset{(p>v')\bullet t}{\Longrightarrow} v}$ output: $\dfrac{e \overset{t}{\Longrightarrow} v}{p\,v' \Leftarrow e \overset{(p<v')\bullet t}{\Longrightarrow} v}$

application: $\dfrac{e_1 \overset{t_1}{\Longrightarrow} \lambda x.e \quad e_2 \overset{t_2}{\Longrightarrow} v' \quad e\{v'/x\} \overset{t_3}{\Longrightarrow} v}{e_1\,e_2 \overset{t_1;t_2;t_3}{\Longrightarrow} v}$

choice: $\dfrac{i_1 \overset{t}{\Longrightarrow} v}{i_1 + i_2 \overset{t}{\Longrightarrow} v}$ $\dfrac{i_2 \overset{t}{\Longrightarrow} v}{i_1 + i_2 \overset{t}{\Longrightarrow} v}$

parallel: $\dfrac{e_1 \overset{t_1}{\Longrightarrow} v \quad e_2 \overset{t_2}{\Longrightarrow} v'}{e_1 \,|\rangle\, e_2 \overset{t_1 \,|\rangle\, t_2}{\Longrightarrow} v}$

Figure 4: The relational semantics.

Definition 4.2 (Trace composition)

$$\begin{aligned} \varepsilon \,;\, t &= t \\ (m \bullet t_1) \,;\, t_2 &= m \bullet (t_1 \,;\, t_2) \\ (t_1 \,|\rangle\, t_2) \,;\, t_3 &= (t_1 \,;\, t_3) \,|\rangle\, t_2 \end{aligned}$$

Theorem 4.1 *Trace composition is associative, i.e.* $t_1 \,;\, (t_2 \,;\, t_3) = (t_1 \,;\, t_2) \,;\, t_3$.

Proof We use structural induction on t_1. Consider the possible forms of this term:

1. $t_1 = \varepsilon$, i.e. we must prove $\varepsilon \,;\, (t_2 \,;\, t_3) = (\varepsilon \,;\, t_2) \,;\, t_3$ which follows trivially from the definition of composition.

2. $t_1 = m \bullet t_1'$, i.e. we must prove $m \bullet t_1' \,;\, (t_2 \,;\, t_3) = (m \bullet t_1' \,;\, t_2) \,;\, t_3$. We deduce that by definition $m \bullet t_1' \,;\, (t_2 \,;\, t_3) = m \bullet (t_1' \,;\, (t_2 \,;\, t_3))$ and also $(m \bullet t_1' \,;\, t_2) \,;\, t_3 = m \bullet ((t_1' \,;\, t_2) \,;\, t_3)$. The terms are therefore equal if we can prove that $t_1' \,;\, (t_2 \,;\, t_3) = (t_1' \,;\, t_2) \,;\, t_3$ and this follows from the inductive hypothesis.

3. $t_1 = t_{11} \,|\rangle\, t_{12}$, i.e. we must prove $(t_{11} \,|\rangle\, t_{12}) \,;\, (t_2 \,;\, t_3) = ((t_{11} \,|\rangle\, t_{12}) \,;\, t_2) \,;\, t_3$. Now $(t_{11} \,|\rangle\, t_{12}) \,;\, (t_2 \,;\, t_3) = (t_{11} \,;\, (t_2 \,;\, t_3)) \,|\rangle\, t_{12}$ (using the definition of composition) and the induction hypothesis allows us to deduce that this is equivalent to $((t_{11} \,;\, t_2) \,;\, t_3) \,|\rangle\, t_{12}$. Applying the definition of composition twice more, in the 'reverse' direction, yields the desired result.

□

Initially we will restrict our attention to just those expressions whose evaluation terminates; we will relax this limitation slightly in Section 5. Figure 4 defines a relational semantics for our (restricted) language, where $e \Longrightarrow v$ is used as an abbreviation for $e \overset{\varepsilon}{\Longrightarrow} v$.[2] The rules do not tell the whole story as they do not deal with the possibility of two subprocesses communicating with each other. If $e \overset{t}{\Longrightarrow} v$ then t records all of the interactions required during the evaluation, not just those that need to be accepted by the environment. For example, suppose that e is the expression

$$(p\, q \Leftarrow p) \,|\rangle\, (p \Rightarrow \lambda x.x)$$

We can use the semantics to deduce that $e \overset{t}{\Longrightarrow} p$ for any trace of the form

$$(p < q \bullet \varepsilon) \,|\rangle\, (p > v \bullet \varepsilon)$$

where v is any value. However, these traces don't reflect the possibility that the result could be produced without any external intervention. To capture this possibility, we define a reduction on traces as follows:

1) $$\frac{}{m \bullet t \xrightarrow{m} t}$$

2) $$\frac{t_1 \xrightarrow{m} t_1' \quad t_2 \xrightarrow{\overline{m}} t_2'}{t_1 \,|\rangle\, t_2 \xrightarrow{\tau} t_1' \,|\rangle\, t_2'}$$

3) $$\frac{t_1 \xrightarrow{\alpha} t_1'}{t_1 \,|\rangle\, t_2 \xrightarrow{\alpha} t_1' \,|\rangle\, t_2}$$

4) $$\frac{t_2 \xrightarrow{\alpha} t_2'}{t_1 \,|\rangle\, t_2 \xrightarrow{\alpha} t_1 \,|\rangle\, t_2'}$$

At this point the reader may well begin to wonder whether we have really achieved anything. After all, the total number of rules defining the 'relational' semantics is now larger than the original transitional semantics (although the use of evaluation contexts makes a direct comparison rather misleading). However, the definition of traces, and their reduction, is unlikely to change when applied to a much larger language. Furthermore, the growth in the number of propagation rules in a transitional semantics (or the number of cases in an evaluation context) easily offsets the four rules defining trace reduction for larger examples. Our approach exploits some of the advantages of both styles of semantics. The relational approach is used to describe the sequential components of the language, whilst the transitional approach is used to describe the ways in which these components interact. It is possible to merge the effects of the trace reductions into the body of the relational rules, thus avoiding a separate set of transitional rules. However, having tried both approaches, we believe that the two level approach leads to a cleaner presentation of the semantics.

[2]The semantics is based on substitutions rather than environments. This was done in order to simplify the proof that the two semantics are indeed equivalent, but has no other significance.

Returning to the plot, we will use $t \longrightarrow^* t'$ as an abbreviation for

$$\exists n \geq 0,\, t_1, \ldots, t_n \cdot t = t_0 \xrightarrow{\tau} t_1 \xrightarrow{\tau} \ldots \xrightarrow{\tau} t_n = t'$$

We will then write $t \xrightarrow{m}{}^* t'$ if there exists t_0, t_1 such that $t \longrightarrow^* t_0$, $t_0 \xrightarrow{m} t_1$, and $t_1 \longrightarrow^* t'$. One of our initial assumptions was that we were only interested in evaluating closed terms to produce a final value. Furthermore, let us for the moment only consider expressions that completely terminate when evaluated. This suggests that we should only consider values in our new semantics that can be produced using a trace t such that $t \longrightarrow^* t'$ and $\mathsf{terminated}(t)$ for some t', where

Definition 4.3 $\mathsf{terminated}(t) = \begin{cases} \mathsf{terminated}(t_1) \wedge \mathsf{terminated}(t_2), & \text{if } t = t_1 \between t_2; \\ t = \varepsilon, & \text{otherwise.} \end{cases}$

This corresponds to the situation where all communication requests are satisfied internally. The following definition captures this intuition.

Definition 4.4 $\mathsf{Eval} \downarrow (e) = \{v : \mathsf{Values} \mid \exists t, t' \cdot e \overset{t}{\Longrightarrow} v \wedge t \longrightarrow^* t' \wedge \mathsf{terminated}(t')\}$

We would expect the set of possible values computed by the current version of our relational semantics to be equal to $\mathsf{eval} \downarrow (e)$. We prove this equality via the two inclusions, starting with $\mathsf{Eval} \downarrow (e) \subseteq \mathsf{eval} \downarrow (e)$. First we need a few auxiliary lemmas.

Lemma 4.2 $\forall i, v, t \cdot\ i \overset{t}{\Longrightarrow} v\ \supset\ \exists m, t' \cdot t = m \bullet t'$

Proof Structural induction on i. □

Lemma 4.3
If $e \overset{t}{\Longrightarrow} v$
then either i) $t = \varepsilon \wedge e \longrightarrow^* v$
or ii) $t = m \bullet t' \wedge \exists e' \cdot e \xrightarrow{m}{}^* e' \wedge e' \overset{t'}{\Longrightarrow} v$
or iii) $t = t_1 \between t_2 \wedge \exists e_1, e_2, v' \cdot e \longrightarrow^* e_1 \between e_2 \wedge e_1 \overset{t_1}{\Longrightarrow} v \wedge e_2 \overset{t_2}{\Longrightarrow} v'$

Proof We use induction on the depth of inference of $e \overset{t}{\Longrightarrow} v$. Consider the possible forms for e:

- $e \equiv p$. The constant rule must have been used, i.e. $t = \varepsilon$ and $v = p$. But then $p \longrightarrow^* p$, as required.
- $e \equiv \lambda x.\, e'$. The constant rule is the only one that could have been used and so we know that e is closed. Furthermore, $t = \varepsilon$ and $v = e$. Trivially $e \longrightarrow^* v$, as required.
- $e \equiv e_1\, e_2$. The application rule must have been used, and so we must have $e_1 \overset{t_1}{\Longrightarrow} \lambda x.\, e'$, $e_2 \overset{t_2}{\Longrightarrow} v'$ and $e'\{v'/x\} \overset{t_3}{\Longrightarrow} v$ for some x, e', v', t_1, t_2, t_3 such that $t = t_1 \fatsemi t_2 \fatsemi t_3$. Consider the possible cases for t:

1. $t \equiv \varepsilon$, i.e. $t_1 = t_2 = t_3 = \varepsilon$. The induction hypothesis allows us to assume that $e_1 \longrightarrow^* \lambda x.\, e'$, $e_2 \longrightarrow^* v'$ and $e'\{v'/x\} \longrightarrow^* v$. The propagation rule may then be used (repeatedly) to deduce that

$$e_1\ e_2 \longrightarrow^* (\lambda x.\, e')\ e_2 \longrightarrow^* (\lambda x.\, e')\ v'$$

   The application rule allows us to reduce this term to $e'\{v'/x\}$ and this is sufficient to show that $e_1\ e_2 \longrightarrow^* v$.

2. $t \equiv m \bullet t'$. This constrains t_1, t_2 and t_3 to one of three cases:
   (a) $t_1 \equiv m \bullet t_1'$, i.e. $t' = t_1' \fatsemi t_2 \fatsemi t_3$. By the induction hypothesis, $\exists e_1' \cdot e_1 \xrightarrow{m}{}^* e_1' \wedge e_1' \overset{t_1'}{\Longrightarrow} \lambda x.\, e'$. The propagation rule may then be used to deduce that $e_1\ e_2 \xrightarrow{m}{}^* e_1'\ e_2$. The application rule can be used to prove that $e_1'\ e_2 \overset{t'}{\Longrightarrow} v$.
   (b) $t_1 \equiv \varepsilon$ and $t_2 \equiv m \bullet t_2'$, i.e. $t' = t_2' \fatsemi t_3$. By the induction hypothesis, $e_1 \longrightarrow^* \lambda x.\, e'$ and $\exists e_2' \cdot e_2 \xrightarrow{m}{}^* e_2' \wedge e_2' \overset{t_2'}{\Longrightarrow} v'$. The propagation rule may then be used to deduce that $e_1\ e_2 \xrightarrow{m}{}^* (\lambda x.\, e')\ e_2'$. The expression $\lambda x.\, e'$ is closed (as $e_1 \overset{t_1}{\Longrightarrow} \lambda x.\, e'$) and so the constant and application rules can then be used to deduce that $(\lambda x.\, e')\ e_2' \overset{t'}{\Longrightarrow} v$.
   (c) $t_1 \equiv \varepsilon$, $t_2 \equiv \varepsilon$ and $t_3 \equiv m \bullet t_3'$, i.e. $t' = t_3'$. By the induction hypothesis, $e_1 \longrightarrow^* \lambda x.\, e'$, $e_2 \longrightarrow^* v'$ and $\exists e'' \cdot e'\{v'/x\} \xrightarrow{m}{}^* e'' \wedge e'' \overset{t'}{\Longrightarrow} v$. Using the propagation rule,

$$e_1\ e_2 \longrightarrow^* (\lambda x.\, e')\ e_2 \longrightarrow^* (\lambda x.\, e')\ v' \longrightarrow e'\{v'/x\} \xrightarrow{m}{}^* e''$$

   as required.

3. $t \equiv t' \mid\rangle\, t''$. Again this constrains t_1, t_2 and t_3 to one of three cases:
   (a) $t_1 \equiv t_{11} \mid\rangle\, t_{12}$. Then $(t_{11} \mid\rangle\, t_{12}) \fatsemi (t_2 \fatsemi t_3) = (t_{11} \fatsemi t_2 \fatsemi t_3) \mid\rangle\, t_{12}$, so $t' = t_{11} \fatsemi t_2 \fatsemi t_3$ and $t'' = t_{12}$. By the induction hypothesis, $\exists e_{11}, e_{12}, v'' \cdot e_1 \longrightarrow^* e_{11} \mid\rangle\, e_{12}$, $e_{11} \overset{t_{11}}{\Longrightarrow} \lambda x.\, e'$ and $e_{12} \overset{t_{12}}{\Longrightarrow} v''$. So

$$e_1\ e_2 \longrightarrow^* (e_{11} \mid\rangle\, e_{12})\ e_2 \longrightarrow (e_{11}\ e_2) \mid\rangle\, e_{12}$$

   Now $e_{11}\ e_2 \overset{t_{11};t_2;t_3}{\Longrightarrow} v$, i.e. $e_{11}\ e_2 \overset{t'}{\Longrightarrow} v$, and $e_{12} \overset{t_{12}}{\Longrightarrow} v''$, i.e. $e_{12} \overset{t''}{\Longrightarrow} v''$, as required.
   (b) $t_1 = \varepsilon$ and $t_2 \equiv t_{21} \mid\rangle\, t_{22}$. Then $t_2 \fatsemi t_3 = (t_{21} \mid\rangle\, t_{22}) \fatsemi t_3 = (t_{21} \fatsemi t_3) \mid\rangle\, t_{22}$, so $t' = t_{21} \fatsemi t_3$ and $t'' = t_{22}$. By the induction hypothesis, $e_1 \longrightarrow^* \lambda x.\, e'$ and $\exists e_{21}, e_{22}, v'' \cdot e_2 \longrightarrow^* e_{21} \mid\rangle\, e_{22}$, $e_{21} \overset{t_{21}}{\Longrightarrow} v'$ and $e_{22} \overset{t_{22}}{\Longrightarrow} v''$. So

$$e_1\ e_2 \longrightarrow^* (\lambda x.\, e')\ (e_{21} \mid\rangle\, e_{22}) \longrightarrow ((\lambda x.\, e')\ e_{21}) \mid\rangle\, e_{22}$$

   Now $(\lambda x.\, e')\ e_{21} \overset{t'}{\Longrightarrow} v$ and $e_{22} \overset{t''}{\Longrightarrow} v''$, as required.

(c) $t_1 = t_2 = \varepsilon$ and $t_3 \equiv t' \,|\rangle\, t''$. By the induction hypothesis, $e_1 \longrightarrow^* \lambda x.e'$, $e_2 \longrightarrow^* v'$ and $\exists e_{31}, e_{32}, v'' \cdot e'\{v'/x\} \longrightarrow^* e_{31} \,|\rangle\, e_{32}$, $e_{31} \stackrel{t'}{\Longrightarrow} v$ and $e_{32} \stackrel{t''}{\Longrightarrow} v''$. So $e_1\, e_2 \longrightarrow^* (\lambda x.e')\, v' \longrightarrow e'\{v'/x\} \longrightarrow^* e_{31} \,|\rangle\, e_{32}$, as required.

- $e \equiv e_1 \,|\rangle\, e_2$. The parallel rule must have been used and so we may conclude that $e_1 \stackrel{t_1}{\Longrightarrow} v$ and $e_2 \stackrel{t_2}{\Longrightarrow} v'$, for some v', where $t = t_1 \,|\rangle\, t_2$ as required.
- $e \equiv a \Rightarrow e'$. The input rule must have been used, i.e. a must be a port p, and so $\exists v', t' \cdot t = (p > v') \bullet t'$ and $e'\, v' \stackrel{t'}{\Longrightarrow} v$. But $(p \Rightarrow e') \stackrel{p>v'}{\longrightarrow} e'\, v'$ as required.
- $e \equiv a\, a' \Leftarrow e'$ The output rule must have been used, i.e. a must be a port p and a' a value v', and so $t = (p < v') \bullet t'$ and $e' \stackrel{t'}{\Longrightarrow} v$ for some t'. But $(p\, v' \Leftarrow e') \stackrel{p<v'}{\longrightarrow} e'$ as required.
- $e \equiv i_1 + i_2$. There are two possibilities depending on which of the choice rules was used. In the first case we can deduce that $i_1 \stackrel{t}{\Longrightarrow} v$. Furthermore, Lemma 4.2 constrains t to be of the form $m \bullet t'$ and the induction hypothesis may then be used to deduce that $\exists e' \cdot i_1 \stackrel{m}{\longrightarrow} e'$ and $e' \stackrel{t'}{\Longrightarrow} v$. But then $i_1 + i_2 \stackrel{m}{\longrightarrow} e'$ as required. The case where the second choice rule was used is similar.

□

Lemma 4.4 $\quad e \stackrel{t}{\Longrightarrow} v \;\wedge\; t \stackrel{\alpha}{\longrightarrow} t' \;\supset\; \exists e' \cdot e \stackrel{\alpha}{\longrightarrow}^* e' \;\wedge\; e' \stackrel{t'}{\Longrightarrow} v$

Proof We use induction on the depth of inference of $t \stackrel{\alpha}{\longrightarrow} t'$. How could $t \stackrel{\alpha}{\longrightarrow} t'$ have been derived? Let's consider the cases:

- $t \equiv m \bullet t'$. From Lemma 4.3 we may deduce $\exists e' \cdot e \stackrel{m}{\longrightarrow}^* e' \;\wedge\; e' \stackrel{t'}{\Longrightarrow} v$ as required.
- $t \equiv t_1 \,|\rangle\, t_2$, $t_1 \stackrel{\alpha}{\longrightarrow} t_1'$ and $t' = t_1' \,|\rangle\, t_2$. From Lemma 4.3 we may deduce $\exists e_1, e_2, v' \cdot e \longrightarrow^* e_1 \,|\rangle\, e_2$, $e_1 \stackrel{t_1}{\Longrightarrow} v$ and $e_2 \stackrel{t_2}{\Longrightarrow} v'$. From the induction hypothesis $\exists e_1' \cdot e_1 \stackrel{\alpha}{\longrightarrow}^* e_1'$ and $e_1' \stackrel{t_1'}{\Longrightarrow} v$. So $e_1 \,|\rangle\, e_2 \stackrel{\alpha}{\longrightarrow}^* e_1' \,|\rangle\, e_2$, i.e. $e \stackrel{\alpha}{\longrightarrow}^* e_1' \,|\rangle\, e_2$ and $e_1' \,|\rangle\, e_2 \stackrel{t'}{\Longrightarrow} v$ as required.
- $t \equiv t_1 \,|\rangle\, t_2$, $t_2 \stackrel{\alpha}{\longrightarrow} t_2'$ and $t' = t_1 \,|\rangle\, t_2'$. From Lemma 4.3 we may deduce $\exists e_1, e_2, v' \cdot e \longrightarrow^* e_1 \,|\rangle\, e_2$, $e_1 \stackrel{t_1}{\Longrightarrow} v$ and $e_2 \stackrel{t_2}{\Longrightarrow} v'$. From the induction hypothesis $\exists e_2' \cdot e_2 \stackrel{\alpha}{\longrightarrow}^* e_2'$ and $e_2' \stackrel{t_2'}{\Longrightarrow} v'$. So $e_1 \,|\rangle\, e_2 \stackrel{\alpha}{\longrightarrow}^* e_1 \,|\rangle\, e_2'$, i.e. $e \stackrel{\alpha}{\longrightarrow}^* e_1 \,|\rangle\, e_2'$ and $e_1 \,|\rangle\, e_2' \stackrel{t'}{\Longrightarrow} v$ as required.

- $t \equiv t_1 \,|\rangle\, t_2$, $t_1 \xrightarrow{m} t_1'$, $t_2 \xrightarrow{\overline{m}} t_2'$, $t' = t_1' \,|\rangle\, t_2'$ and $\alpha = \tau$. From Lemma 4.3 we may deduce $\exists e_1, e_2, \nu' \cdot e \longrightarrow^* e_1 \,|\rangle\, e_2$, $e_1 \overset{t_1}{\Longrightarrow} \nu$ and $e_2 \overset{t_2}{\Longrightarrow} \nu'$. From the induction hypothesis $\exists e_1' \cdot e_1 \xrightarrow{m}{}^* e_1'$ and $e_1' \overset{t_1'}{\Longrightarrow} \nu$. Furthermore $\exists e_2' \cdot e_2 \xrightarrow{\overline{m}}{}^* e_2'$ and $e_2' \overset{t_2'}{\Longrightarrow} \nu'$. So $e_1 \,|\rangle\, e_2 \xrightarrow{\tau}{}^* e_1' \,|\rangle\, e_2'$ by appropriate interleavings of the transitions. As $e_1' \,|\rangle\, e_2' \overset{t'}{\Longrightarrow} \nu$ we are finished.

□

Lemma 4.5 $\quad e \overset{t}{\Longrightarrow} \nu \;\wedge\; \mathsf{terminated}(t) \;\supset$
$\exists e' \cdot e \longrightarrow^* e' \;\wedge\; \mathsf{main}(e') = \nu \;\wedge\; \mathsf{terminated}(e')$

Proof Use structural induction on t. Suppose $e \overset{t}{\Longrightarrow} \nu \;\wedge\; \mathsf{terminated}(t)$. Consider the cases for t:

- $t = \varepsilon$. Then the result follows from Lemma 4.3.
- $t = t_1 \,|\rangle\, t_2$, where $\mathsf{terminated}(t_1)$ and $\mathsf{terminated}(t_2)$. From Lemma 4.3 $\exists e_1, e_2, \nu' \cdot e \longrightarrow^* e_1 \,|\rangle\, e_2$, $e_1 \overset{t_1}{\Longrightarrow} \nu$ and $e_2 \overset{t_2}{\Longrightarrow} \nu'$. From the induction hypothesis, $\exists e_1' \cdot e_1 \longrightarrow^* e_1'$ and $\mathsf{main}(e_1') = \nu$ and $\mathsf{terminated}(e_1')$. Furthermore, $\exists e_2' \cdot e_2 \longrightarrow^* e_2'$ and $\mathsf{main}(e_2') = \nu'$ and $\mathsf{terminated}(e_2')$. But then $e_1 \,|\rangle\, e_2 \longrightarrow^* e_1' \,|\rangle\, e_2'$, $\mathsf{main}(e_1' \,|\rangle\, e_2') = \nu$ and $\mathsf{terminated}(e_1' \,|\rangle\, e_2')$.

□

We can now state and prove one half of the equivalence result.

Theorem 4.6 $\quad \forall e \cdot \mathsf{Eval}\downarrow(e) \subseteq \mathsf{eval}\downarrow(e)$

Proof Suppose that $\nu \in \mathsf{Eval}\downarrow(e)$, i.e. $\exists t, t' \cdot e \overset{t}{\Longrightarrow} \nu$, $t \longrightarrow^* t'$ and $\mathsf{terminated}(t')$. We must prove that $\exists e' \cdot e \longrightarrow^* e'$, $\mathsf{main}(e') = \nu$ and $\mathsf{terminated}(e')$. We do this by induction on the length of the transition sequence $t \longrightarrow^* t'$. In the base case, where the sequence is of zero length, we know that t is equal to t'. Lemma 4.5 may be used to show that $e \longrightarrow^* e'$ for some e' such that $\mathsf{main}(e') = \nu$ and $\mathsf{terminated}(e')$. Consider now the case where $t \longrightarrow^* t'$ is inferred using at least one step. There must exist a t'' such that $t \longrightarrow t'' \longrightarrow^* t'$. From Lemma 4.4 we know $\exists e'' \cdot e \longrightarrow^* e''$ and $e'' \overset{t''}{\Longrightarrow} \nu$. From the induction hypothesis we know $\exists e' \cdot e'' \longrightarrow^* e'$, $\mathsf{main}(e') = \nu$ and $\mathsf{terminated}(e')$. □

We now prove the other direction, i.e. $\mathsf{eval}\downarrow(e) \subseteq \mathsf{Eval}\downarrow(e)$, starting with some auxiliary lemmas.

Lemma 4.7 $\quad t \xrightarrow{\alpha} t' \supset \forall t'' \cdot t \mathbin{;} t'' \xrightarrow{\alpha} t' \mathbin{;} t''$

Proof We use induction on the depth of inference of $t \xrightarrow{\alpha} t'$:

- $t \equiv m \bullet t'$, where $\alpha = m$. Then $(m \bullet t') \mathbin{;} t'' = m \bullet (t' \mathbin{;} t'') \xrightarrow{m} t' \mathbin{;} t''$
- $t \equiv t_1 \mid\rangle t_2$, $t_1 \xrightarrow{\alpha} t_1'$, $t' = t_1' \mid\rangle t_2$. By the induction hypothesis, $t_1 \mathbin{;} t'' \xrightarrow{\alpha} t_1' \mathbin{;} t''$ so $(t_1 \mathbin{;} t'') \mid\rangle t_2 \xrightarrow{\alpha} (t_1' \mathbin{;} t'') \mid\rangle t_2$, i.e. $(t_1 \mid\rangle t_2) \mathbin{;} t'' \xrightarrow{\alpha} (t_1' \mid\rangle t_2) \mathbin{;} t''$.
- $t \equiv t_1 \mid\rangle t_2$, $t_2 \xrightarrow{\alpha} t_2'$, $t' = t_1 \mid\rangle t_2'$.
  Then $(t_1 \mathbin{;} t'') \mid\rangle t_2 \xrightarrow{\alpha} (t_1 \mathbin{;} t'') \mid\rangle t_2'$, i.e. $(t_1 \mid\rangle t_2) \mathbin{;} t'' \xrightarrow{\alpha} (t_1 \mid\rangle t_2') \mathbin{;} t''$.
- $t \equiv t_1 \mid\rangle t_2$, $t_1 \xrightarrow{m} t_1'$, $t_2 \xrightarrow{\overline{m}} t_2'$ and $t' = t_1' \mid\rangle t_2'$. By the induction hypothesis, $t_1 \mathbin{;} t'' \xrightarrow{m} t_1' \mathbin{;} t''$ so $(t_1 \mathbin{;} t'') \mid\rangle t_2 \xrightarrow{\tau} (t_1' \mathbin{;} t'') \mid\rangle t_2'$, i.e. $(t_1 \mid\rangle t_2) \mathbin{;} t'' \xrightarrow{\tau} (t_1' \mid\rangle t_2') \mathbin{;} t''$.

□

Lemma 4.8 $\quad e \xrightarrow{m} e' \wedge e' \overset{t}{\Longrightarrow} \nu \supset \exists t' \cdot e \overset{t'}{\Longrightarrow} \nu \wedge t' \xrightarrow{m} t$

Proof We use induction on the depth of inference of $e \xrightarrow{m} e'$.

- $(p \Rightarrow e) \xrightarrow{p>\nu'} e\,\nu'$ and $e\,\nu' \overset{t}{\Longrightarrow} \nu$. Using the input rule we may deduce that $(p \Rightarrow e) \overset{(p>\nu')\bullet t}{\Longrightarrow} \nu$. Furthermore, $(p > \nu') \bullet t \xrightarrow{p>\nu'} t$.
- $(p\,\nu' \Leftarrow e) \xrightarrow{p<\nu'} e$ and $e \overset{t}{\Longrightarrow} \nu$. Using the output rule we may deduce that $(p\,\nu' \Leftarrow e) \overset{(p<\nu')\bullet t}{\Longrightarrow} \nu$. Furthermore, $(p < \nu') \bullet t \xrightarrow{p<\nu'} t$.
- $(i_1 + i_2) \xrightarrow{m} e$, $i_1 \xrightarrow{m} e$ and $e \overset{t}{\Longrightarrow} \nu$. By the induction hypothesis $\exists t' \cdot i_1 \overset{t'}{\Longrightarrow} \nu$ and $t' \xrightarrow{m} t$. The choice rule then allows us to deduce that $i_1 + i_2 \overset{t'}{\Longrightarrow} \nu$. The case where i_2 makes the transition is similar.
- $e_1 \xrightarrow{m} e_1'$, $e_1\,e_2 \xrightarrow{m} e_1'\,e_2 \overset{t}{\Longrightarrow} \nu$. The application rule must have been used to deduce $e_1'\,e_2 \overset{t}{\Longrightarrow} \nu$ and so we must have $e_1' \overset{t_1}{\Longrightarrow} \lambda x.e$, $e_2 \overset{t_2}{\Longrightarrow} \nu'$ and $e\{\nu'/x\} \overset{t_3}{\Longrightarrow} \nu$ for some t_1, t_2, t_3, x, e, ν', where $t = t_1 \mathbin{;} t_2 \mathbin{;} t_3$. By the induction hypothesis, $\exists t_1' \cdot e_1 \overset{t_1'}{\Longrightarrow} \lambda x.e$ and $t_1' \xrightarrow{m} t_1$. So $e_1\,e_2 \overset{t_1';t_2;t_3}{\Longrightarrow} \nu$. But by Lemma 4.7 $t_1' \mathbin{;} t_2 \mathbin{;} t_3 \xrightarrow{m} t_1 \mathbin{;} t_2 \mathbin{;} t_3 = t$.
- $e_2 \xrightarrow{m} e_2'$, $(\lambda x.e)\,e_2 \xrightarrow{m} (\lambda x.e)\,e_2' \overset{t}{\Longrightarrow} \nu$. The application rule must have been used and so we must have $e_2' \overset{t_2}{\Longrightarrow} \nu'$ and $e\{\nu'/x\} \overset{t_3}{\Longrightarrow} \nu$ for some t_2, t_3, ν', where $t = t_2 \mathbin{;} t_3$. By the induction hypothesis, $\exists t_2' \cdot e_2 \overset{t_2'}{\Longrightarrow} \nu'$ and $t_2' \xrightarrow{m} t_2$. So $(\lambda x.e)\,e_2 \overset{t_2';t_3}{\Longrightarrow} \nu$. But by Lemma 4.7 $t_2' \mathbin{;} t_3 \xrightarrow{m} t_2 \mathbin{;} t_3$.

- $e_1 \xrightarrow{m} e_1'$, $e_1 \between e_2 \xrightarrow{m} e_1' \between e_2 \stackrel{t}{\Longrightarrow} \nu$. The parallel rule must have been used and so we must have $t = t_1 \between t_2$ and $\exists \nu' \cdot e_1' \stackrel{t_1}{\Longrightarrow} \nu$ and $e_2 \stackrel{t_2}{\Longrightarrow} \nu'$. By the induction hypothesis, $\exists t_1' \cdot e_1 \stackrel{t_1'}{\Longrightarrow} \nu$ and $t_1' \xrightarrow{m} t_1$. So $e_1 \between e_2 \stackrel{t_1' \between t_2}{\Longrightarrow} \nu$. But $t_1' \between t_2 \xrightarrow{m} t_1 \between t_2$.
- $e_2 \xrightarrow{m} e_2'$, $e_1 \between e_2 \xrightarrow{m} e_1 \between e_2' \stackrel{t}{\Longrightarrow} \nu$. The parallel rule must have been used and so we must have $t = t_1 \between t_2$ and $\exists \nu' \cdot e_1 \stackrel{t_1}{\Longrightarrow} \nu$ and $e_2' \stackrel{t_2}{\Longrightarrow} \nu'$. By the induction hypothesis, $\exists t_2' \cdot e_2 \stackrel{t_2'}{\Longrightarrow} \nu'$ and $t_2' \xrightarrow{m} t_2$. So $e_1 \between e_2 \stackrel{t_1 \between t_2'}{\Longrightarrow} \nu$. But $t_1 \between t_2' \xrightarrow{m} t_1 \between t_2$. □

The next lemma is similar, except that it allows e to move silently to e'. An unfortunate complication is that we must demand that e is a closed term. Usually this would be implicit due to the assumption that $e' \stackrel{t}{\Longrightarrow} \nu$. However, the presence of the choice operator, and its associated semantic rules, allows non-closed terms to be evaluated as long as the free variables are in the discarded choice branches. We could remedy this by altering the choice rules (e.g. by having an additional premise evaluating the discarded branch). However, introducing an explicit constraint seems simpler.

Lemma 4.9 $\quad closed(e) \wedge e \xrightarrow{\tau} e' \wedge e' \stackrel{t}{\Longrightarrow} \nu \supset \exists t' \cdot e \stackrel{t'}{\Longrightarrow} \nu \wedge t' \longrightarrow^* t$

Proof We use induction on the depth of inference of $e \xrightarrow{\tau} e'$.

- $e = e_1 \between e_2 \longrightarrow e_1' \between e_2'$, $e_1 \xrightarrow{m} e_1'$, $e_2 \xrightarrow{\overline{m}} e_2'$, $e_1' \between e_2' \stackrel{t}{\Longrightarrow} \nu$. This last transition must have been deduced using the Parallel rule, and so $\exists \nu' \cdot e_1' \stackrel{t_1}{\Longrightarrow} \nu$ and $e_2' \stackrel{t_2}{\Longrightarrow} \nu'$, where $t = t_1 \between t_2$. We can then use Lemma 4.8 to deduce that $\exists t_1', t_2' \cdot e_1 \stackrel{t_1'}{\Longrightarrow} \nu$, $e_2 \stackrel{t_2'}{\Longrightarrow} \nu'$, $t_1' \xrightarrow{m} t_1$ and $t_2' \xrightarrow{\overline{m}} t_2$, i.e. $t_1' \between t_2' \xrightarrow{\tau} t_1 \between t_2$. But then $e_1 \between e_2 \stackrel{t_1' \between t_2'}{\Longrightarrow} \nu$, so take t' to be $t_1' \between t_2'$.
- $e = (\lambda x.e')\, \nu' \longrightarrow e'\{\nu'/x\}$ and $e'\{\nu'/x\} \stackrel{t}{\Longrightarrow} \nu$. Now $\lambda x.e' \Longrightarrow \lambda x.e'$ (as e is closed), and $\nu' \Longrightarrow \nu'$ so $(\lambda x.e')\, \nu' \stackrel{t}{\Longrightarrow} \nu$. Furthermore, $t \longrightarrow^* t$.
- $e = (\lambda x.e')\, e_1 \between e_2 \longrightarrow ((\lambda x.e')\, e_1) \between e_2 \stackrel{t}{\Longrightarrow} \nu$. The Parallel rule must have been used so $\exists t_1, t_2, \nu' \cdot (\lambda x.e')\, e_1 \stackrel{t_1}{\Longrightarrow} \nu$ and $e_2 \stackrel{t_2}{\Longrightarrow} \nu'$, where $t = t_1 \between t_2$. But then $\exists t_{11}, t_{12}, \nu'' \cdot e_1 \stackrel{t_{11}}{\Longrightarrow} \nu''$ and $e'\{\nu''/x\} \stackrel{t_{12}}{\Longrightarrow} \nu$, where $t_1 = t_{11} \fatsemi t_{12}$. So $(\lambda x.e')\, e_1 \between e_2 \stackrel{t_{11} \between t_2 ; t_{12}}{\Longrightarrow} \nu$. Now $(t_{11} \between t_2) \fatsemi t_{12} = (t_{11} \fatsemi t_{12}) \between t_2 = t$.
- $e = e_1 \between e_2\, e_3 \longrightarrow (e_1\, e_3) \between e_2 \stackrel{t}{\Longrightarrow} \nu$. So $t = t_1 \between t_2$, $(e_1\, e_3) \stackrel{t_1}{\Longrightarrow} \nu$ and $e_2 \stackrel{t_2}{\Longrightarrow} \nu'$ for some t_1, t_2, ν'. From this we can deduce that $e_1 \stackrel{t_{11}}{\Longrightarrow} \lambda x.e'$, $e_3 \stackrel{t_{12}}{\Longrightarrow} \nu''$ and $e'\{\nu''/x\} \stackrel{t_{13}}{\Longrightarrow} \nu$ for some t_{11}, t_{12}, t_{13}, x, e', ν'' such that $t_1 = t_{11} \fatsemi t_{12} \fatsemi t_{13}$. But then $(e_1 \between e_2)\, e_3 \stackrel{(t_{11} \between t_2) ; t_{12} ; t_{13}}{\Longrightarrow} \nu$. However,
$(t_{11} \between t_2) \fatsemi t_{12} \fatsemi t_{13} = ((t_{11} \fatsemi t_{12}) \between t_2) \fatsemi t_{13} = (t_{11} \fatsemi t_{12} \fatsemi t_{13}) \between t_2 = t$.

- $e = e_1\ e_2 \longrightarrow e_1'\ e_2 \stackrel{t}{\Longrightarrow} \nu$, i.e. $e_1 \longrightarrow e_1'$. There must exist t_1, t_2, t_3, x, e', ν' such that $e_1' \stackrel{t_1}{\Longrightarrow} \lambda x.e'$, $e_2 \stackrel{t_2}{\Longrightarrow} \nu'$ and $e'\{\nu'/x\} \stackrel{t_3}{\Longrightarrow} \nu$, where $t = t_1 \mathbin{;} t_2 \mathbin{;} t_3$. Then by the induction hypothesis (as e_1 must be closed), $\exists t_1' \cdot e_1 \stackrel{t_1'}{\Longrightarrow} \lambda x.e'$ and $t_1' \longrightarrow^* t_1$. So $e_1\ e_2 \stackrel{t_1';t_2;t_3}{\Longrightarrow} \nu$. But from Lemma 4.7 we can deduce that $t_1' \mathbin{;} t_2 \mathbin{;} t_3 \longrightarrow^* t_1 \mathbin{;} t_2 \mathbin{;} t_3 = t$.

- $e = (\lambda x.e')\ e_2 \longrightarrow (\lambda x.e')\ e_2' \stackrel{t}{\Longrightarrow} \nu$, i.e. $e_2 \longrightarrow e_2'$. There must exist t_2, t_3, ν' such that $e_2' \stackrel{t_2}{\Longrightarrow} \nu'$ and $e'\{\nu'/x\} \stackrel{t_3}{\Longrightarrow} \nu$, where $t = t_2 \mathbin{;} t_3$. Then by the induction hypothesis (as e_2 must be closed), $\exists t_2' \cdot e_2 \stackrel{t_2'}{\Longrightarrow} \nu'$ and $t_2' \longrightarrow^* t_2$. So $(\lambda x.e')\ e_2 \stackrel{t_2';t_3}{\Longrightarrow} \nu$. But from Lemma 4.7 we can deduce that $t_2' \mathbin{;} t_3 \longrightarrow^* t_2 \mathbin{;} t_3 = t$.

- $e = e_1 \mid\rangle e_2 \longrightarrow e_1' \mid\rangle e_2 \stackrel{t}{\Longrightarrow} \nu$, i.e. $e_1 \longrightarrow e_1'$. There must exist t_1, t_2, ν' such that $e_1' \stackrel{t_1}{\Longrightarrow} \nu$, $e_2 \stackrel{t_2}{\Longrightarrow} \nu'$, where $t = t_1 \mid\rangle t_2$. Then by the induction hypothesis (as e_1 closed), $\exists t_1' \cdot e_1 \stackrel{t_1'}{\Longrightarrow} \nu$ and $t_1' \longrightarrow^* t_1$. So $e_1 \mid\rangle e_2 \stackrel{t_1' \mid\rangle t_2}{\Longrightarrow} \nu$. But $t_1' \mid\rangle t_2 \longrightarrow^* t_1 \mid\rangle t_2 = t$.

- $e = e_1 \mid\rangle e_2 \longrightarrow e_1 \mid\rangle e_2' \stackrel{t}{\Longrightarrow} \nu$, i.e. $e_2 \longrightarrow e_2'$. There must exist t_1, t_2, ν' such that $e_1 \stackrel{t_1}{\Longrightarrow} \nu$, $e_2' \stackrel{t_2}{\Longrightarrow} \nu'$, where $t = t_1 \mid\rangle t_2$. Then by the induction hypothesis (as e_2 closed), $\exists t_2' \cdot e_2 \stackrel{t_2'}{\Longrightarrow} \nu'$ and $t_2' \longrightarrow^* t_2$. So $e_1 \mid\rangle e_2 \stackrel{t_1 \mid\rangle t_2'}{\Longrightarrow} \nu$. But $t_1 \mid\rangle t_2' \longrightarrow^* t_1 \mid\rangle t_2 = t$.

□

Lemma 4.10 $\mathsf{terminated}(e) \supset \exists \nu, t \cdot e \stackrel{t}{\Longrightarrow} \nu$ *and* $\mathsf{terminated}(t)$.

Proof Structural induction on e. □

Lemma 4.11 $\mathit{closed}(e) \wedge e \stackrel{\alpha}{\longrightarrow} e' \supset \mathit{closed}(e')$.

Proof Induction on the depth of inference of $e \stackrel{\alpha}{\longrightarrow} e'$. □

We can now prove the second half of our main theorem, namely

Theorem 4.12 $\forall$ *closed* $e \cdot \mathsf{eval}\downarrow(e) \subseteq \mathsf{Eval}\downarrow(e)$

Proof Suppose that $\nu \in \mathsf{eval}\downarrow(e)$, i.e. $\exists e' \cdot e \longrightarrow^* e'$, $\mathsf{main}(e') = \nu$ and $\mathsf{terminated}(e')$. We must prove that $\exists t, t' \cdot e \stackrel{t}{\Longrightarrow} \nu$, $t \longrightarrow^* t'$ and $\mathsf{terminated}(t')$. We use induction on the length of the sequence $e \longrightarrow^* e'$:

- The sequence is of length 0, i.e. $e = e'$, $\mathsf{main}(e) = \nu$ and $\mathsf{terminated}(e)$. We use structural induction on e:

1. $e = e_1 \parallel e_2$, i.e. $\mathsf{main}(e) = \mathsf{main}(e_1) = \nu$, $\mathsf{terminated}(e_1)$ and $\mathsf{terminated}(e_2)$. Furthermore, both e_1 and e_2 are closed. By the induction hypothesis, $\exists t, t' \cdot e_1 \stackrel{t}{\Longrightarrow} \nu$, $t \longrightarrow^* t'$ and $\mathsf{terminated}(t')$. We may use Lemma 4.10 to deduce that $\exists \nu', t_2 \cdot e_2 \stackrel{t_2}{\Longrightarrow} \nu'$ and $\mathsf{terminated}(t_2)$. Therefore $e_1 \parallel e_2 \stackrel{t \parallel t_2}{\Longrightarrow} \nu$ and $t \parallel t_2 \longrightarrow^* t' \parallel t_2$, where $\mathsf{terminated}(t' \parallel t_2)$.
2. $e = \nu$. Then trivially $e \Longrightarrow \nu$ using the constant rule.

- The sequence is of length > 0, i.e. $\exists e'' \cdot e \longrightarrow e'' \longrightarrow^* e'$. By the induction hypothesis, using Lemma 4.11 to deduce that e'' is closed, $\exists t, t' \cdot e'' \stackrel{t}{\Longrightarrow} \nu$, $t \longrightarrow^* t'$ and $\mathsf{terminated}(t')$. From Lemma 4.9 we may deduce that $e \stackrel{t'}{\Longrightarrow} \nu$ and $t' \longrightarrow^* t$ for some t' as required.

□

This concludes our proof that for all closed expressions e $\mathsf{Eval} \downarrow (e) = \mathsf{eval} \downarrow (e)$. The result might be criticised as being rather weak because it only deals with terminating programs, and doesn't address other aspects of a program's behaviour such as deadlock. We now turn our attention to these issues.

5 Nontermination

Consider the program $e \equiv p \parallel \uparrow$, where $\uparrow$ is used as an abbreviation for the non-terminating expression $(\lambda x.\, x\, x)\, (\lambda x.\, x\, x)$. Clearly $\mathsf{main}(e) = p$ and so $\mathsf{eval}(e) = \{p\}$. This follows from our view that any remaining evaluation, even a non-terminating one, is irrelevant once the main thread of the execution has been reduced to a value. Unfortunately it is easy to show that there exists no ν, t such that $e \stackrel{t}{\Longrightarrow} \nu$ and so $\mathsf{Eval} \downarrow (e) = \{\}$. We would like to define something analogous to eval using the relational semantics. Non-terminating computations give rise to infinite proof trees, and infinite traces, in our current setting. One approach might be to use the coinduction/maximal fix point interpretation of the rules in these cases. Alternatively we could add some additional rules to the semantics to 'prune' the unwanted parts of the proof tree, i.e. the parts of the proof that have no influence on the final value. This is the approach followed here. We introduce an element $\perp$, disjoint from the set of values, and define $\nu_\perp \in \mathsf{Values}_\perp = \mathsf{Values} \cup \{\perp\}$. The relation $\longrightarrow\!\!\!\!\!\rightarrow$ will now be a subset of $\mathsf{Expressions} \times \mathsf{Traces} \times \mathsf{Values}_\perp$. Informally, we use $\perp$ to stand for an unevaluated result, and so our first rule is very simple:

$$\frac{}{e \Longrightarrow \perp}$$

i.e. we don't have to evaluate anything! This will allow us to prune the evaluation of expressions from the proof tree once the value of the main thread has been determined. We must also propagate $\perp$ through compound expressions, as in

$$\frac{e_1 \stackrel{t}{\Longrightarrow} \perp}{e_1\, e_2 \stackrel{t}{\Longrightarrow} \perp}$$

This approach may result in a large number of such propagation rules. However, in a language such as ML, with an exception mechanism, we can use the existing propagation rules for this purpose. We just need to make $\bot$ an 'exception' that cannot be caught. Our simple language doesn't have exceptions, and so we add the propagation rules explicitly for $\bot$. The complete set of rules are presented in Figure 5.

Returning to the example presented at the beginning of this section, it is simple to show that $p \mid\rangle \uparrow \overset{\varepsilon \mid\rangle \varepsilon}{\Longrightarrow} p$ using the modified set of rules:

$$\frac{p \overset{\varepsilon}{\Longrightarrow} p \quad \uparrow \overset{\varepsilon}{\Longrightarrow} \bot}{p \mid\rangle \uparrow \overset{\varepsilon \mid\rangle \varepsilon}{\Longrightarrow} p}$$

A further example may be helpful at this point. Let $e \equiv \lambda x.\, p\, q \Leftarrow x\, x$. Here is one possible evaluation of the program $(p \Rightarrow (\lambda y.\, y)) \mid\rangle (e\, e)$:

$$\frac{\dfrac{\dfrac{\vdots}{(\lambda y.\, y)\, q \Longrightarrow q}}{(p \Rightarrow (\lambda y.\, y)) \overset{p>q\bullet\varepsilon}{\Longrightarrow} q} \qquad \frac{\overline{e \Longrightarrow e} \quad \overline{e \Longrightarrow e} \quad \dfrac{\overline{e\, e \Longrightarrow \bot}}{(p\, q \Leftarrow e\, e) \overset{p<q\bullet\varepsilon}{\Longrightarrow} \bot}}{(e\, e) \overset{p<q\bullet\varepsilon}{\Longrightarrow} \bot}}{(p \Rightarrow (\lambda y.\, y)) \mid\rangle (e\, e) \overset{(p>q\bullet\varepsilon) \mid\rangle (p<q\bullet\varepsilon)}{\Longrightarrow} q}$$

In our definition of Eval $\downarrow$ we demanded that all communications must be satisfied internally, i.e. $\exists t' \cdot t \longrightarrow^* t'$ and terminated(t'). If we wish to mimic eval then this requirement is too strong. We should only demand that the communications performed by the main thread are satisfied internally. This motivates the following definitions:

Definition 5.1 $$\mathsf{main}(t) = \begin{cases} \mathsf{main}(t_1), & \text{if } t = t_1 \mid\rangle t_2; \\ m \bullet \mathsf{main}(t'), & \text{if } t = m \bullet t'; \\ \varepsilon, & \text{otherwise.} \end{cases}$$

Clearly terminated(t) implies main$(t) = \varepsilon$ for all t.

Definition 5.2 $$\mathsf{synchronised}(t) = \exists t' \cdot t \longrightarrow^* t' \wedge \mathsf{main}(t') = \varepsilon$$

Definition 5.3 $$\mathsf{Eval}(e) = \{v : \mathsf{Values} \mid \exists t, t' \cdot e \overset{t}{\Longrightarrow} v \wedge \mathsf{synchronised}(t)\}$$

These definitions are sufficient to capture the meaning of eval, as demonstrated by the following theorem.

Theorem 5.1 $\forall$ *closed* $e \cdot \mathsf{eval}(e) = \mathsf{Eval}(e)$

Proof Similar in structure to Theorems 4.6 and 4.12 and their associated lemmas. □

constant: $\dfrac{}{v \Longrightarrow v}$

input: $\dfrac{e\,v' \overset{t}{\Longrightarrow} v_\perp}{p \Rightarrow e \overset{(p>v')\bullet t}{\Longrightarrow} v_\perp}$ output: $\dfrac{e \overset{t}{\Longrightarrow} v_\perp}{p\,v' \Leftarrow e \overset{(p<v')\bullet t}{\Longrightarrow} v_\perp}$

application: $\dfrac{e_1 \overset{t_1}{\Longrightarrow} \lambda x.e \quad e_2 \overset{t_2}{\Longrightarrow} v' \quad e\{v'/x\} \overset{t_3}{\Longrightarrow} v_\perp}{e_1\,e_2 \overset{t_1;t_2;t_3}{\Longrightarrow} v_\perp}$

$\dfrac{e_1 \overset{t}{\Longrightarrow} \perp}{e_1\,e_2 \overset{t}{\Longrightarrow} \perp}$ $\dfrac{e_1 \overset{t_1}{\Longrightarrow} \lambda x.e \quad e_2 \overset{t_2}{\Longrightarrow} \perp}{e_1\,e_2 \overset{t_1;t_2}{\Longrightarrow} \perp}$

parallel: $\dfrac{e_1 \overset{t_1}{\Longrightarrow} v_\perp \quad e_2 \overset{t_2}{\Longrightarrow} v'_\perp}{e_1 \,|\rangle\, e_2 \overset{t_1 \,|\rangle\, t_2}{\Longrightarrow} v_\perp}$

choice: $\dfrac{i_1 \overset{t}{\Longrightarrow} v_\perp}{i_1 + i_2 \overset{t}{\Longrightarrow} v_\perp}$ $\dfrac{i_2 \overset{t}{\Longrightarrow} v_\perp}{i_1 + i_2 \overset{t}{\Longrightarrow} v_\perp}$

undefined: $\dfrac{}{e \Longrightarrow \perp}$

Figure 5: The relational semantics allowing non-termination.

Suppose we have a trace t that can be silently reduced to t', i.e. $t \longrightarrow^* t'$, such that $\mathsf{main}(t') = \varepsilon$. The trace t' may contain actions in branches off of the main thread, and so $\mathsf{terminated}(t')$ will usually be false. However, we can imagine pruning these actions from t, leaving the main thread unchanged, so that the corresponding reduced thread is terminated. Suppose that $\exists t, t' \cdot e \stackrel{t}{\Longrightarrow} \nu_\perp \wedge t \longrightarrow^* t'$ and $\mathsf{main}(t') = \varepsilon$. We should be able to 'prune' t to produce a trace t_p such that $e \stackrel{t_p}{\Longrightarrow} \nu_\perp$, $t_p \longrightarrow^* t'_p$ and $\mathsf{terminated}(t'_p)$. If we could do this then a consequence would be that $\mathsf{Eval}(e) = \mathsf{Eval} \downarrow (e)$ for the extended semantics.

How can we formalise this notion of pruning? We start by defining a prefix relation on traces, written $t_1 \leq t_2$.

Definition 5.4 (Trace prefixes)

$$\begin{array}{rcl} \varepsilon & \leq & \varepsilon \\ \varepsilon & \leq & m \bullet t \\ m \bullet t_1 & \leq & m \bullet t_2 \ \mathit{iff}\ t_1 \leq t_2 \\ t_1 \,|\rangle\, t_2 & \leq & t'_1 \,|\rangle\, t'_2 \ \mathit{iff}\ t_1 \leq t'_1 \ \wedge \ t_2 \leq t'_2 \end{array}$$

We would like to prove that if $e \stackrel{t}{\Longrightarrow} \nu_\perp$, $t' \leq t$ and $\mathsf{main}(t) = \mathsf{main}(t')$ then $e \stackrel{t'}{\Longrightarrow} \nu_\perp$.

Lemma 5.2 $\quad t \leq t_1 \fatsemi t_2 \supset \exists t'_1, t'_2 \cdot t = t'_1 \fatsemi t'_2 \ \wedge \ t'_1 \leq t_1 \ \wedge \ t'_2 \leq t_2$.

Proof We use structural induction on t_1. Consider the possible forms of this trace:

1. $t_1 = \varepsilon$, i.e. $t \leq t_2$. We just need to choose ε for t'_1 and t for t'_2.
2. $t_1 = m \bullet t_{11}$, i.e. $t_1 \fatsemi t_2 = m \bullet (t_{11} \fatsemi t_2)$. So t must be either
   (a) ε. Then take t'_1 and t'_2 to be ε.
   (b) $m \bullet t'$ for some t' such that $t' \leq t_{11} \fatsemi t_2$. By the I.H., there exists t''_1, t''_2 such that $t' = t''_1 \fatsemi t''_2$ where $t''_1 \leq t_{11}$ and $t''_2 \leq t_2$. But then $t = (m \bullet t''_1) \fatsemi t''_2$ so take t'_1 to be $m \bullet t''_1$ and t'_2 to be t''_2.
3. $t_1 = t_{11} \,|\rangle\, t_{12}$, i.e. $t \leq (t_{11} \fatsemi t_2) \,|\rangle\, t_{12}$. So t must be of the form $t' \,|\rangle\, t''$ where $t' \leq t_{11} \fatsemi t_2$ and $t'' \leq t_{12}$. By the I.H., there exists t'_1, t'_2 such that $t' = t'_1 \fatsemi t'_2$ where $t'_1 \leq t_{11}$ and $t'_2 \leq t_2$. So $t = t' \,|\rangle\, t'' = (t'_1 \fatsemi t'_2) \,|\rangle\, t'' = (t'_1 \,|\rangle\, t'') \fatsemi t'_2$. We must show that $(t'_1 \,|\rangle\, t'') \leq (t_{11} \,|\rangle\, t_{12}$ which follows from the definition of $\leq$.

□

A simple extension of this proof allows us to conclude that if $\mathsf{main}(t'_1) \neq \mathsf{main}(t_1)$ then $t'_2 = \varepsilon$.

Definition 5.5 $\quad t \leq_m t' \stackrel{\text{def}}{=} t \leq t' \ \wedge \ \mathsf{main}(t) = \mathsf{main}(t')$

If the pruned trace, t, contains the same main thread as $t_1 \fatsemi t_2$ then this constrains t'_1 and t'_2 as follows:

Lemma 5.3 $t \leq_m t_1 \mathbin{;} t_2 \supset \exists t_1', t_2' \cdot t = t_1' \mathbin{;} t_2' \wedge t_1' \leq_m t_1 \wedge t_2' \leq_m t_2.$

If a trace t_1 can make a transition to t_2 and t_2 can be pruned then there should be a corresponding pruned version of t_1. This is formalised in the following lemma:

Lemma 5.4 $t_1 \xrightarrow{\alpha} t_2 \wedge t_2' \leq t_2 \supset \exists t_1' \leq t_1 \cdot t_1' \xrightarrow{\alpha} t_2'$

Proof We use induction on the depth of inference of $t_1 \xrightarrow{\alpha} t_2$. Consider the different ways in which this could have been proved:

1. $t_1 = m \bullet t_2$, $\alpha = m$. Just take t_1' to be $m \bullet t_2'$.
2. $t_1 = t_{11} \between t_{12}$, $t_{11} \xrightarrow{\alpha} t_{21}$, $t_2 = t_{21} \between t_{12}$. Then there exists $t_{21}' \leq t_{21}$, $t_{22}' \leq t_{12}$ such that $t_2' = t_{21}' \between t_{22}'$. Using the I.H. we can deduce that there exists $t_{11}' \leq t_{11}$ such that $t_{11}' \xrightarrow{\alpha} t_{21}'$. Now $t_{11}' \between t_{22}' \leq t_1$ and $t_{11}' \between t_{22}' \xrightarrow{\alpha} t_{21}' \between t_{22}' = t_2'$ and so we can take t_1' to be $t_{11}' \between t_{22}'$.
3. The case where t_{12} makes the move is symmetrical to the previous case.
4. $t_1 = t_{11} \between t_{12}$, $t_{11} \xrightarrow{m} t_{21}$, $t_{12} \xrightarrow{\overline{m}} t_{22}$, $\alpha = \tau$, $t_2 = t_{21} \between t_{22}$. Then there exists $t_{21}' \leq t_{21}$, $t_{22}' \leq t_{22}$ such that $t_2' = t_{21}' \between t_{22}'$. Using the I.H. we can deduce that there exists $t_{11}' \leq t_{11}$ such that $t_{11}' \xrightarrow{m} t_{21}'$ and $t_{12}' \leq t_{12}$ such that $t_{12}' \xrightarrow{\overline{m}} t_{22}'$. But then $t_{11}' \between t_{12}' \xrightarrow{\tau} t_{21}' \between t_{22}' = t_2'$. Furthermore, $t_{11}' \between t_{12}' \leq t_1$ so take t_1' to be $t_{11}' \between t_{12}'$

□

Corollary 5.5 $t_1 \longrightarrow^* t_2 \wedge t_2' \leq t_2 \supset \exists t_1' \leq t_1 \cdot t_1' \longrightarrow^* t_2'$

These lemmas can be easily extended to the corresponding $\leq_m$ cases:

Lemma 5.6 $t_1 \xrightarrow{\alpha} t_2 \wedge t_2' \leq_m t_2 \supset \exists t_1' \leq_m t_1 \cdot t_1' \xrightarrow{\alpha} t_2'$

Corollary 5.7 $t_1 \longrightarrow^* t_2 \wedge t_2' \leq_m t_2 \supset \exists t_1' \leq_m t_1 \cdot t_1' \longrightarrow^* t_2'$

The following theorem captures our intuition about trace pruning. The proof follows from the previous lemmas.

Theorem 5.8

If $t \longrightarrow^* t' \wedge \mathsf{main}(t') = \varepsilon$
then $\exists t_1, t_1' \cdot t_1 \leq_m t \wedge t_1 \longrightarrow^* t_1' \wedge \mathsf{terminated}(t_1')$

Suppose we have an evaluation of the form $e \overset{t}{\Longrightarrow} v$ such that $\mathsf{synchronised}(t)$, i.e. $t \longrightarrow^* t'$ and $\mathsf{main}(t') = \varepsilon$ for some t'. From our previous theorem we can deduce that $\exists t_1, t_1' \cdot t_1 \leq_m t \wedge t_1 \longrightarrow^* t_1' \wedge \mathsf{terminated}(t_1')$. If we can prove that $e \overset{t_1}{\Longrightarrow} v$ then this would show that $v \in \mathsf{Eval} \downarrow (e)$, i.e. $\mathsf{Eval} \downarrow = \mathsf{Eval}$ for the extended semantics. Intuitively we would expect this to be true as v only depends on the main thread of the execution, which, in turn, should only depend on the main thread of t. The following theorem confirms that this is indeed the case.

Theorem 5.9

If $e \stackrel{t}{\Longrightarrow} v_\perp$ and $t' \leq t$

then either i) $\mathsf{main}(t) = \mathsf{main}(t')$ and $e \stackrel{t'}{\Longrightarrow} v_\perp$

or ii) $\mathsf{main}(t) \neq \mathsf{main}(t')$ and $e \stackrel{t'}{\Longrightarrow} \perp$

Proof Induction on $e \stackrel{t}{\Longrightarrow} v_\perp$.

1. $e \equiv v$, $v \Longrightarrow v$, so t' must be ε, $\mathsf{main}(t') = \mathsf{main}(t)$

2. $e \equiv p \Rightarrow e'$, $t = (p > v') \bullet t''$, $e'\, v' \stackrel{t''}{\Longrightarrow} v_\perp$. Now if $t' \leq t$ then $t' = (p > v') \bullet t'''$, $t''' \leq t''$. If $\mathsf{main}(t''') = \mathsf{main}(t'')$ then by the I.H. $e'\, v' \stackrel{t'''}{\Longrightarrow} v_\perp$ and so $p \Rightarrow e' \stackrel{t'}{\Longrightarrow} v_\perp$ which is what is expected as $\mathsf{main}(t') = \mathsf{main}(t)$. If $\mathsf{main}(t''') \neq \mathsf{main}(t'')$ then by the I.H. $e'\, v' \stackrel{t'''}{\Longrightarrow} \perp$. Then $p \Rightarrow e' \stackrel{t'}{\Longrightarrow} \perp$ which is what is expected as $\mathsf{main}(t') \neq \mathsf{main}(t)$.

3. The output case is similar.

4. $e \equiv e_1\, e_1$, $e_1 \stackrel{t_1}{\Longrightarrow} \lambda x.e'$, $e_2 \stackrel{t_2}{\Longrightarrow} v'$, $e'\{v'/x\} \stackrel{t_3}{\Longrightarrow} v_\perp$ and $t = t_1 \fatsemi t_2 \fatsemi t_3$. From Lemma 5.2, $t' = t'_1 \fatsemi t'_2 \fatsemi t'_3$, $t'_1 \leq t_1$, $t'_2 \leq t_2$, $t'_3 \leq t_3$ for some t'_1, t'_2, t'_3. Suppose $\mathsf{main}(t) = \mathsf{main}(t')$. Then from Lemma 5.3 $t'_1 \leq_m t_1$, $t'_2 \leq_m t_2$ and $t'_3 \leq_m t_3$. Therefore, by the I.H., $e_1\, e_2 \stackrel{t'}{\Longrightarrow} v_\perp$. The situation is slightly more complex if $\mathsf{main}(t) \neq \mathsf{main}(t')$. It must be the case that either $\mathsf{main}(t_1) \neq \mathsf{main}(t'_1)$ or $\mathsf{main}(t_2) \neq \mathsf{main}(t'_2)$ or $\mathsf{main}(t_3) \neq \mathsf{main}(t'_3)$. Consider these cases separately:

   (a) $\mathsf{main}(t_1) \neq \mathsf{main}(t'_1)$. From the extension to Lemma 5.2 we can conclude that $t'_2 = t'_3 = \varepsilon$. By the I.H., $e_1 \stackrel{t'_1}{\Longrightarrow} \perp$ and so $e_1\, e_2 \stackrel{t'}{\Longrightarrow} \perp$.

   (b) $\mathsf{main}(t_1) = \mathsf{main}(t'_1)$ and $\mathsf{main}(t_2) \neq \mathsf{main}(t'_2)$. From the extension to Lemma 5.2 we can conclude that $t'_3 = \varepsilon$. By the I.H., $e_1 \stackrel{t'_1}{\Longrightarrow} \lambda x.e'$ and $e_2 \stackrel{t'_2}{\Longrightarrow} \perp$ and so $e_1\, e_2 \stackrel{t'}{\Longrightarrow} \perp$.

   (c) $\mathsf{main}(t_1) = \mathsf{main}(t'_1)$, $\mathsf{main}(t_2) = \mathsf{main}(t'_2)$ and $\mathsf{main}(t_3) \neq \mathsf{main}(t'_3)$. By the I.H., $e_1 \stackrel{t'_1}{\Longrightarrow} \lambda x.e'$, $e_2 \stackrel{t'_2}{\Longrightarrow} v'$ and $e'\{v'/x\} \stackrel{t'_3}{\Longrightarrow} \perp$ and so $e_1\, e_2 \stackrel{t'}{\Longrightarrow} \perp$.

5. $e \equiv e_1\, e_2$, $e_1 \stackrel{t}{\Longrightarrow} \perp$. The result follows from the I.H..

6. $e \equiv e_1\, e_2$, $e_1 \stackrel{t_1}{\Longrightarrow} \lambda x.e'$, $e_2 \stackrel{t_2}{\Longrightarrow} \perp$, $t = t_1 \fatsemi t_2$. Then by Lemma 5.2, $t' = t'_1 \fatsemi t'_2$ for some $t'_1 \leq t_1$, $t'_2 \leq t_2$. Suppose $\mathsf{main}(t) = \mathsf{main}(t')$. Then $\mathsf{main}(t'_1) = \mathsf{main}(t_1)$ and $\mathsf{main}(t'_2) = \mathsf{main}(t_2)$ so $e_1\, e_2 \stackrel{t'}{\Longrightarrow} \perp$. If $\mathsf{main}(t) \neq \mathsf{main}(t')$ then either

   (a) $\mathsf{main}(t'_1) \neq \mathsf{main}(t_1)$ and by Lemma 5.2 $t'_2 = \varepsilon$. By the I.H. $e_1 \stackrel{t'_1}{\Longrightarrow} \perp$ and so $e_1\, e_2 \stackrel{t'}{\Longrightarrow} \perp$, or

(b) $\mathsf{main}(t_1') = \mathsf{main}(t_1)$ and $\mathsf{main}(t_2') = \mathsf{main}(t_2)$. By the I.H. $e_1 \stackrel{t_1'}{\Longrightarrow} \lambda x.e'$ and $e_2 \stackrel{t_2'}{\Longrightarrow} \bot$, hence $e_1\ e_2 \stackrel{t'}{\Longrightarrow} \bot$.

7. $e \equiv e_1 \mid\rangle e_2$, $e_1 \stackrel{t_1}{\Longrightarrow} v_\bot$, $e_2 \stackrel{t_2}{\Longrightarrow} v'_\bot$, $t = t_1 \mid\rangle t_2$. There must exist t_1', t_2' such that $t' = t_1' \mid\rangle t_2'$ and $t_1' \leq t_1, t_2' \leq t_2$. Furthermore $\mathsf{main}(t) = \mathsf{main}(t_1)$ and $\mathsf{main}(t') = \mathsf{main}(t_1')$. Suppose $\mathsf{main}(t_1) = \mathsf{main}(t_1')$. By the I.H. $e_1 \stackrel{t_1'}{\Longrightarrow} v_\bot$ and $e_2 \stackrel{t_2'}{\Longrightarrow} v''_\bot$ (where $v''_\bot$ is either $v''_\bot$ or $\bot$) so $e_1 \mid\rangle e_2 \stackrel{t'}{\Longrightarrow} v_\bot$. If $\mathsf{main}(t_1) \neq \mathsf{main}(t_1')$ then by the I.H. $e_1 \stackrel{t_1'}{\Longrightarrow} \bot$, e_2 evaluation as before, and so $e_1 \mid\rangle e_2 \stackrel{t'}{\Longrightarrow} \bot$.

8. Choice is easy as is

9. the undefined case as $t' = t = \varepsilon$.

□

The transitional semantics uses two definitions of evaluation, one for the case where all processes must terminate and the other for the case where only the main thread must terminate. The above result shows that for the relational case one definition suffices, but we must alter the semantics rules instead to capture the required intention.

6 Divergence and Deadlock

Even if we are content with our closed world assumptions about program evaluation, there are other aspects of behaviour we might wish to observe. For example, we would like to know whether the evaluation can diverge. Divergence is simple to define in the transitional case. However, it is far from clear how we can give an account of divergence in terms of the relational semantics. Consider the evaluation of the two programs

$$e_1 = (\lambda x.x\ x)\ (\lambda x.x\ x) \quad \text{and} \quad e_2 = p\ p$$

In both cases they evaluate to $\bot$ in the modified semantics, and have no evaluation at all in the original. However, informally, e_1 diverges and e_2 represents an illegal program. One manifestation of this problem arises when we consider type soundness proofs. We would like to say that if an expression e can be type-checked then we cannot get stuck at run-time. However, because the distinction between divergence and getting stuck is masked in the relational style of semantics we often settle for a weaker result, namely that *if* e produces a value v, then v is of the computed type. To do better than this requires the separation of the stuck and diverging cases. One approach involves the identification of all stuck configurations and the generation of error values in these cases. These error values then need to be propagated, just as we did for $\bot$ [10]. Needless to say, such extensions quickly obscure the underlying

semantics, and diminish the advantages of the relational style. Deadlock is another example of getting stuck, and causes similar problems for this style of semantics.

These problems are present in all relational semantics, not just those for concurrent languages. The relational semantics for SML [7] has ably demonstrated that it is not necessary for a semantic description to capture all aspects of a program's behaviour for it to be useful. Ideally we would like a range of semantics for a programming language, each contributing something to our understanding of program behaviour. There should be compatible definitions for those concepts that are expressible in more than one semantics, as was the case for the evaluation function in our example. The user is then free to choose whichever semantics allows him to reason about a particular aspect of program behaviour in the simplest way.

7 Conclusions

We have shown that it is indeed possible to extend the relational style of semantics to describe concurrent computations. We can view the result as combining the advantages of both approaches. The sequential aspects of the computation are concisely expressed by the relational rules whilst process synchronisation is described by a small transition system.

A number of issues warrant further investigation. Can we propagate errors and exceptions using the traces? We can't detect deadlock in the trace reduction semantics because of the choice rules. However, if we kept more information in the traces, such as the alternatives rejected at the choice points, then this might allow us to talk about deadlock in this framework. We also need to apply our technique to some more realistic examples, such as Facile [8], LCS [2], and the various concurrent extensions of SML (e.g. CML [9]) before our claims of conciseness can be completely vindicated.

References

[1] D. Berry. *Generating Program Animators from Programming Language Semantics*, CS Thesis, University of Edinburgh, 1991.

[2] B. Berthomieu. *Implementing CCS, the LCS Experiment*, Technical Report 89425, LAAS-CNRS, 1989.

[3] G. Kahn. *Natural Semantics*, In Programming of Future Generation Computers, pages 237-258, North-Holland Publishing Company, 1988.

[4] R. Milner. *Communication and Concurrency*, Prentice Hall, 1989.

[5] R. Milner. *Functions as processes*, INRIA Technical Report 1154, 1990.

[6] G.D. Plotkin. *Structural Operational Semantics*, Lecture notes, DAIMI FN-19, Aarhus University, Denmark, 1981 (reprinted 1991).

[7] R. Milner, M. Tofte and R. Harper. *The Definition of Standard ML*, MIT Press, 1990.

[8] S. Prasad, A. Giacalone and P. Mishra. *Operational and Algebraic Semantics for Facile: A Symmetric Integration of Concurrent and Functional Programming*, Automata, Languages and Programming (ICALP), LNCS 443, Springer, 1990.

[9] J. Reppy. *CML: A Higher-order Concurrent Language*, ACM Conference on Programming Language Design and Implementation, 1991.

[10] M. Tofte. *Operational Semantics and Polymorphic Type Inference*, CS Thesis, University of Edinburgh, 1988.

[11] A. Wright and M. Felleisen. *A Syntactic Approach to Type Soundness*, Rice Technical Report TR91-160, 1992.