

UNIVERSITY OF SUSSEX

is given in terms of a reduction relation between *configurations* — i.e. sets of  $\lambda_{cv}$ -closed expressions or programs — unfortunately this operational semantics is not compositional in that the behaviour of a  $\lambda_{cv}$ -expression or indeed configuration is not determined by that of its constituents.

Here we give a compositional operational semantics in terms of a labelled transition system for  $\mu$ CML programs — this is not only describes the evaluation steps of programs as in [1] but also the communication potential in terms of the ability to input and output values along communication channels.

We then proceed to demonstrate the usefulness of this compositional operational semantics by using it to define a version of *weak observational equivalence* [2] suitable for  $\mu$ CML. We prove that modulo the usual problems associated with the closure operator of CC, our chosen equivalences preserved by a  $\mu$ CML contexts and therefore may be used as the basis for reasoning about CML programs. In this paper we do not investigate in detail the resulting theory but concentrate on pointing out some of its salient features: for example standard denotational semantics one would expect of a calculus are given and we also show that certain algebraic laws common to process algebras [3] do

we now explain in more detail the contents of the remainder of the paper.

**SECTION 2** we describe the language  $\mu$ CML, a subset of CML. It is a typed language with base types for channels, booleans and integers and type constructors for pairs, functions and delayed computations. These are called Event types. It has the standard constructs and constants associated with the base types and with pairs and functions. In addition it has a selection of the CML constructs and constants for spawning delayed computations: `spawn` gener-



fst	$A$	$B$	$A$	transmit <sub><math>A</math></sub>	chan	$A$	unit event
snd	$A$	$B$	$B$	receive <sub><math>A</math></sub>	chan	$A$	event
add	int	int	int	choose	$A$ event	$A$ event	$A$ event
mul	int	int	int	spawn	(unit	unit)	unit
leq	int	int	bool	wrap	$A$ event	( $A$ $B$ )	$B$ event
sync	$A$ event	$A$		never	unit	$A$ event	
always	$A$	$A$ event					

FIG. 4 E



*William Ferreira, Matthew Hennessy and Alan Jeffrey*

Since  $\mathbf{A}v$  is a constant  $v$  we have

$$\overline{\mathbf{A}v} = v$$

We can construct  $\text{choose } e$  as a choice between *delayed computations* as  $\text{choose}$  as the type  $\mathbf{A} \text{event} \ \mathbf{A} \text{event} \ \mathbf{A} \text{event}$ . To interpret it we introduce a new choice constructor  $ge \ \text{ge}_2$  where  $ge$  and  $ge_2$  are guarded expressions of the same type. When  $\text{choose } e$  proceeds by evaluating  $e$  until it can produce a value  $w$  it must be of the form  $[ge], [ge_2]$  and the evaluation continues by constructing the *delayed computation*  $[ge \ \text{ge}_2]$  as is represented by the rule

$$\frac{e \xrightarrow{[ge], [ge_2]} e}{\text{choose } e \xrightarrow{} e \ [ge \ \text{ge}_2]}$$

The notation introduced is unfortunate as it is used to represent the *internal choice* between processes whereas here it represents *external choice*: we have the following auxiliary rules which are the same as CC substitution

$$\frac{ge \xrightarrow{\alpha} e}{ge \ \text{ge}_2 \xrightarrow{\alpha} e} \quad \frac{ge_2 \xrightarrow{\alpha} e}{ge \ \text{ge}_2 \xrightarrow{\alpha} e}$$

This ends our informal description of the  $\mathbf{A}$  substitution

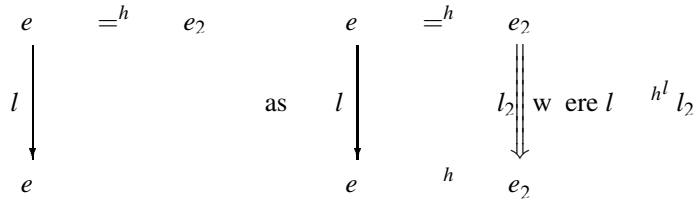




For any purposes strong bisimulation is too fine an equivalence as it is sensitive to the number of reductions performed by expressions. This means that not even the fundamental elementary properties of  $\beta$  reduction such as  $Id = \eta$  where  $Id$  denotes the identity function  $(\lambda x. x)$  are required for weaker *weak bisimulation* which allows  $\tau$  actions to be ignored.

It is in turn required so the more notation. Let  $\equiv^\varepsilon$  be the reflexive transitive closure of  $\rightarrow^\tau$  and let  $\equiv^l$  be  $\equiv^\varepsilon \rightarrow^l$  where any sequence of silent actions followed by an  $l$  action. Note that we are *not* allowing silent actions after the  $l$  action. Let  $\equiv^l$  be  $\equiv^\varepsilon$  if  $l = \tau$  and  $\equiv^l$  otherwise. Then  $\mathcal{R}$  is a *first-order weak simulation* iff it is structure preserving and the following diagram can be completed.

$e$

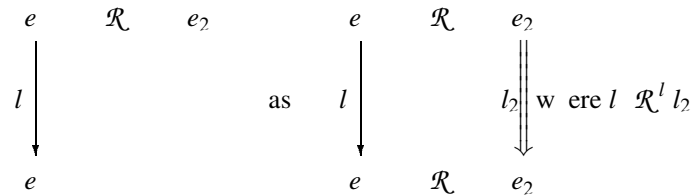


PROPOSITION 1.  $\stackrel{=^h}{=}$  is an equivalence.

PROOF. Similar to the proof of Proposition 1.  $\square$

It is attempted to show, however, since the notion of a process and not the first moves of any processes in its transitions is used above  $\mu\text{CML}$ , counterexamples for  $\stackrel{=^h}{=}$  being a congruence also apply to  $\stackrel{=^h}{=}$ . Its failure was first noted by Rosen [2] for CHOC.

Rosen's solution to this problem is to require that  $\tau$  moves can always be matched by at least one  $\tau$  move which produces a definition of an *irreflexive simulation* as a structure preserving relation where the following diagram can be completed.



Let  $\stackrel{i}{=}$  be the largest reflexive bisimulation.

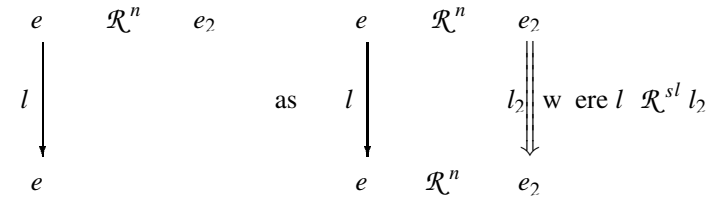
PROPOSITION 2.  $\stackrel{i}{=}$  is a congruence.

PROOF. The proof that  $\stackrel{i}{=}$  is an equivalence is similar to the proof of Proposition 1. The proof that it is a congruence is similar to the proof of Lemma 4.2 in the next section.  $\square$

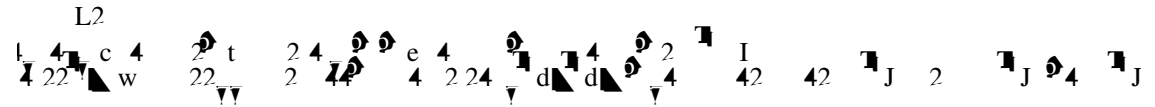
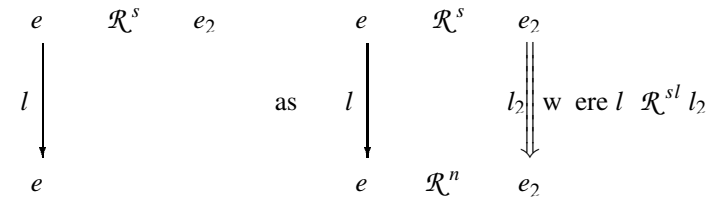
However, this relation is rather too strong for many purposes. For example,  $\text{add}(x, 2) \stackrel{i}{=} \text{add}(x, \text{add}(x, 2))$  since the first can perform more  $\tau$  moves than the second. This is similar to the problem in CHOC where  $a.\tau.P \stackrel{i}{=} a.P$ .

In order to find an appropriate definition of bisimulation for  $\mu\text{CML}$ , we observe that  $\mu\text{CML}$  on *guarded expressions* can be used on *guarded expressions* and not on arbitrary expressions. We can thus ignore the  $\tau$  moves of a process *except* for guarded expressions. For this reason, we have to provide *two* equivalences: one on terms where we are not interested in the  $\tau$  moves and one on terms where we are.

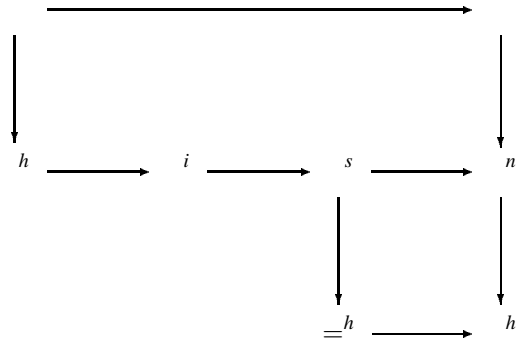
A pair of closed type indexed relations  $\mathcal{R} = (\mathcal{R}^n, \mathcal{R}^s)$  for a *hereditary simulation* we call  $\mathcal{R}^n$  an *insensitive simulation* and  $\mathcal{R}^s$  a *sensitive simulation* iff  $\mathcal{R}^s$ 's structure preserving and we can complete the following diagrams.



and



conclusions:



PROOF For each conclusion show that the first block is a subsequence of the second for of blocks. It is sufficient to show that the conclusions are strictly weaker using the following examples

```

(fn x add( ,2)) h (fn x add(2, ))
let x = in x
choose(receive k, tau(receive k)) i h tau(receive k)
add( ,2) s i add( , add( , ))
n s let x = in x
never() h n tau(never())
h=h let x = in x
  
```

where

```
tau = fn x wrap(always x, sync)
```

Note that this settles an open question [2] of [1] on semantics as to whether it is

and space  $\mathcal{R}$

✦

*William Ferreira, Matthew Hennessy and Alan Jeffrey*

refinement  $\hat{\mathcal{R}}$  be defined

$$\hat{\mathcal{R}}^n = \{(D_n[e], D_n$$



**PROPOSITION 4** *If  $\mathcal{R}$  is an equivalence then  $\mathcal{R}^*$  is symmetric.*

**PROOF** A variant of the proof in [1].

It suffices to show that if  $e \mathcal{R}^* f$  then  $f \mathcal{R}^* e$  and that if  $e \mathcal{R}^* f$  then  $f \mathcal{R}^* e$  we show by induction on  $e$ . If  $e \mathcal{R}^* f$  then either

- $e = D[e] \widehat{\mathcal{R}}^s D[f] \mathcal{R}^s f$  and  $e_i \mathcal{R}^* f_i$  so by induction on  $f_i \mathcal{R}^* e_i$  so  $f \widehat{\mathcal{R}}^s D[f] D \widehat{\mathcal{R}}^s [e] = e$  or
- $e = \text{fix}(x = \text{fn } y \ e) \widehat{\mathcal{R}}^s \text{fix}(x = \text{fn } y \ f) \mathcal{R}^s f$  and  $e \mathcal{R}^* f$  so by induction on  $f \mathcal{R}^* e$  so  $f \widehat{\mathcal{R}}^s \text{fix}(x = \text{fn } y \ f) \mathcal{R}^s \text{fix}(x = \text{fn } y \ e) = e$

The proof for  $\mathcal{R}^n$  is similar.  $\square$

We can use this result to show that  $\cdot$  is a bisimulation.

**PROPOSITION 4** *When restricted to closed expressions of  $\mu\text{CML}^+$ ,  $\cdot$  is a hereditary bisimulation.*

**PROOF** By Proposition 4.4  $\cdot$  is a hereditary simulation and so  $\cdot$  is a hereditary simulation. By Proposition 4  $\cdot$  is symmetric and so  $\cdot$  is a hereditary bisimulation.  $\square$

This gives us the result we set out to prove.

**THEOREM 4**  *$\widehat{\mathcal{R}}^s$  is a congruence, and  $\mathcal{R}^n$  is an uneventful congruence.*

**PROOF** From Proposition 4.4  $\cdot$  is a hereditary bisimulation so  $\widehat{\mathcal{R}}^s$  and by Proposition 4.2  $\cdot$  so  $\cdot$  and  $\widehat{\mathcal{R}}^s$  are the same relation. Hence  $\widehat{\mathcal{R}}^s$  we have the desired result by Proposition 4.  $\square$

## 5 Properties of Weak Bisimulation

In this section we show some results about program equivalence up to hereditary weak bisimulation. One of these equivalences are easy to show but some are trickier and require properties about the transition systems generated by  $\mu\text{CML}^+$ . Although much remains to be done on elaborating the algebraic theory of  $\mu\text{CML}$  programs we open the results in this section and categorize the equivalences can form the basis of a useful theory which generalises those associated with process algebras and functional programming.

We have given an operational semantics to  $\mu\text{CML}$  by extending it with new constructs. Most of which correspond to constructs found in standard process algebras. These include a choice operator, a parallel operator and substitutions of input and output prefixes. The prefixes in  $\mu\text{CML}^{cv}$  have a

signature:  $\text{syntax terms in CC are given as}$

$$\begin{array}{l} \text{CCS prefix } \mu\text{CML}^{cv} \text{ equivalent} \\ k.x.P \quad k \quad \text{fn } x \quad P \\ k.v.P \quad k \quad v \quad \text{fn } x \quad P \\ \tau.P \quad \mathbf{A}() \quad \text{fn } x \quad P \end{array}$$

we now examine the extent to which the choice and parallel operators from a process algebra

can be substituted for the following and hence they are sensitive bisimilar.

$$\begin{array}{l} \Lambda \ e \ e \\ (e \ e_2) \ e \ e \ (e_2 \ e) \\ (e \ e_2) \ e \ (e_2 \ e) \ e \end{array}$$

This satisfies any of the standard laws associated with a parallel operator in a process algebra. However this is not in general symmetric because of its interaction with the product of values.

$$v \ e \ e$$

For example

$$\Lambda \ \Lambda \ \Lambda$$

This means that we can view the parallel composition of processes as being of the form

$$\left( \parallel_i e_i \right) f$$

where the order of the  $e_i$  is unimportant. Note that it is important with respect to rightmost expression in a parallel composition since this is the one that is read of computationally and so can return a value while none of the other expressions can. The choice operator of  $\mu\text{CML}^+$  also satisfies the expected laws from process algebras. The use of a computational monoid that our theory can only be applied to guarded expressions.

$$\begin{array}{l} \Lambda \ ge \ ge \\ (ge \ ge_2) \ ge \ ge \ (ge_2 \ ge) \\ ge \ ge_2 \ ge_2 \ ge \end{array}$$

This means that we can view the sum of guarded expressions as being of the form

$$\bigoplus_i ge_i$$

where the order of the  $ge_i$  is unimportant

In fact guarded expressions can be viewed in a manner quite similar to the *sum forms* used in the development of the algebraic theory of CCS. We can find basic results for the following and hence they are sensible basic results

$$(ge_1 ge_2) \nu \quad (ge_1 \nu) \quad (ge_2 \nu)$$

$$ge \text{ fn } x \quad x \text{ }^s ge$$

$$\mathbf{A} \nu \text{ }^s \mathbf{A}() \text{ fn } x \quad \nu$$

From this we can show by structural induction on terms that all guarded expressions are of a given form

$$ge \text{ }^s \bigoplus_i ge_i ge$$





$\lambda_{cv}$  express ons Instead of ut sets we use *configurations* of  $\mu\text{CML}^{cv}$  express ons given by the gra ar

$$C \text{ Conf} = e \mid C \mid C \mid \Lambda$$

Note that con gurat ons are restr cted for s of  $\mu\text{CML}^+$  express ons. This s w fac tate the co par son between the two se ant cs snce t can be carr ed out for con gurat ons rat er than  $\mu\text{CML}$  express ons.

The se ant cs of  $\Delta$  s expressed as a reduct on re at on = between con gurat ons and reduct ons ave four ndependent sources. The rst nvo ves a sequent a reduct on w t n an nd v dua  $\mu\text{CML}$  express on and t s n turn s de ned us ng anot er reduct on re at on — t e second s t e spawn ng of new *computation threads* w c resu ts n an ncrease n t e nu ber of co ponents of t e con gurat on t e t rd s co un cat on between two express ons and t e ast s requ red to and t e **always** construct e need not at on for eac of t ese and we cons der t e n turn.

The operat ona ru es for sequent a reduct on are de ned *in context* n t e sty e of r g t and Fe e sen  $\Delta$  and t e contexts t at per t reduct on are given by the fo ow ng gra ar

$$E = [\cdot] \mid Ee \mid vE \mid cE \mid (E, e) \mid (v, E) \mid \text{let } x = E \text{ in } e \mid \text{if } E \text{ then } e \text{ else } e$$

The re at on — s de ned to be t e east re at on sat sfy ng t e fo ow ng ru es

$$\begin{aligned} E[cv] &- E[\delta(cv)] & (c \in \{\text{spawn, sync}\}) & \text{const} \\ E[(\text{fix}(x = \text{fn } y \text{ } e))v] &- E[e[\text{fix}(x = \text{fn } y \text{ } e)/x][v/y]] & & \text{beta} \\ E[\text{let } x = v \text{ in } e] &- E[e[v/x]] & & \text{et} \\ E[(v, w)] &- E[v, w] & & \text{par} \end{aligned}$$

Here eac ru e corresponds to a bas c co putat on step n a sequent a ca by va ue language es ou d po nt out t at t e ast ru e does not appear n  $\Delta$  t s p c t n eppy s state nt t e syntact cc ass of t e ter  $(v_1, v_2)$  s e t er *Exp* or *Val* t s a b gu ty s reso ved n favour of *Val* e ave ade t e gra ar una b guous and ave added an exp c t reduct on ru e for reso v ng a b gu ty.

Note t at t e de nt on of — s not co pos tona t e reduct ons of an express on are not de ned n ter s of t e reduct ons of ts sub express ons. The fo ow ng Lemma w be usefu n ater proofs and s ows t at we can recover co pos tona ty.

LEMMA 1. *If  $e \rightarrow e$  then  $E[e] \rightarrow E[e]$ .*

PROOF By exa nat on of t e proof of t e trans t on  $e \rightarrow e$ . □

To capture reduct ons w c nvo ve co un cat on t s necessary to de ne a

tion  $\lambda$  as the  $\mu\text{CML}^+$  semantics and we now compare them. In order to do this we extract a labelled transition system from the  $\mu\text{CML}^{cv}$  semantics by defining

$$C \xrightarrow{\tau} C \text{ iff } C = C$$

$$C \xrightarrow{v} C \text{ iff } C = C \text{ } v \text{ and } C = C \text{ } \Lambda \text{ up to associativity and } \Lambda \text{ left unit}$$

$$C \xrightarrow{k}^v C \text{ iff } C \text{ } k = C \text{ } v$$

$$C \xrightarrow{k}^x C \text{ iff } C \text{ } k \text{ } x = C \text{ } ()$$

We then show that this labelled transition system is weakly bisimilar to the  $\mu\text{CML}^+$  semantics.

**THEOREM 2** *The  $\mu\text{CML}^{cv}$  semantics of a configuration is weakly bisimilar to its  $\mu\text{CML}^+$  semantics.*

The remainder of this section is devoted to proving this result. Although the style of presentation of these two semantics are very different the resulting relations are very similar and there are essentially only two sources for the differences.

The first is that certain reductions in  $\mu\text{CML}^{cv}$  when encoded in the  $\mu\text{CML}^+$  semantics require an additional source of reduction. A typical example is the reduction

$$(\text{fn } x \text{ } e)v \text{ } - \text{ } e[v/x].$$

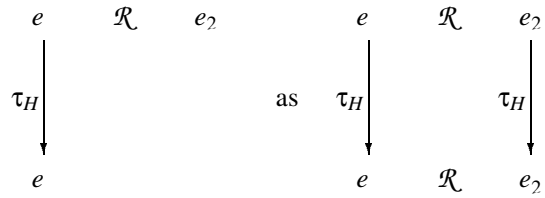
In  $\mu\text{CML}^+$  this requires two reductions

$$(\text{fn } x \text{ } e)v \text{ } - \text{ } \text{let } x = v \text{ in } e \text{ } - \text{ } e[v/x]$$

This problem is addressed by defining the set of source reduction rules such as the second reduction above within the  $\mu\text{CML}^+$  semantics. These turn out to be very simple and we can work with source reduction rules for spawning without further source reduction rules can be added.

The second divergence between the semantics concerns the treatment of spawn expressions in  $\mu\text{CML}^+$  may spawn new processes which give rise to

It is equivalent to a strong first-order bisimulation which respects housekeeping transitions at a relation  $\mathcal{R}$  where we can complete the diagram



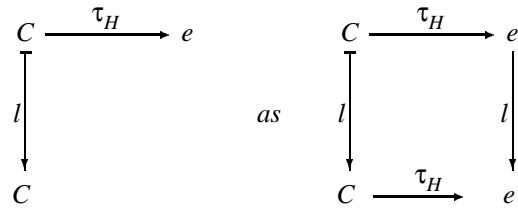
and similarly for  $\mathcal{R}^{-1}$ .

**Proposition 10** *is a strong first-order bisimulation which respects housekeeping.*

**Proof** See the Appendix. □

We can also show a very strong correspondence between reductions of  $\mu\text{CML}^{cv}$  configurations and their denotational counterparts

**Proposition 11** *If  $C \xrightarrow{\tau_H} e$  and  $e$  is tidy, then the following diagrams can be completed:*



and:





could conceivably be necessary to adopt the *context bisimulation equivalence* originally developed in [1]. In short although these theories are being developed independently for these languages any of the techniques developed will be more generally applicable.

## Appendix

This section is devoted to the proof of Proposition 1 and Proposition 2. But first we need some auxiliary results. The following three Propositions state



William Ferreira, Matthew Hennessy and Alan Jeffrey

- 4 M Hennessy *Algebraic Theory of Processes* MIT Press
- C A Hoare *Communicating Sequential Processes* Prentice Hall
- 9 Loren Hoare PFL A functional language for parallel programming In *Proc. Declarative Programming Workshop* pages 4
- 4 Douglas Howe Equational theory of computation systems In *Proc. LICS 89* pages 2
- 4 Douglas Howe Proving congruence of substitution in functional languages [pub](#)  
submitted manuscript 2
- 4 Alan Jeffrey A fully abstract semantics for a concurrent functional language with nondeterminism