Syntactic Control of Concurrency ⋆

D. R. Ghica, A. S. Murawski and C.-H. L. Ong

Oxford University Computing Laboratory
Wolfson Building, Parks Road, Oxford OX1 3QD, UK

Abstract

We consider a finitary procedural programming language (finite data-types, no recursion) extended with parallel composition and binary semaphores. Having first shown that may-equivalence of second-order open terms is undecidable we set out to find a framework in which decidability can be regained with minimum loss of expressivity. To that end we define an annotated type system that controls the number of concurrent threads created by terms and give a fully abstract game semantics for the notion of equivalence induced by typable terms and contexts. Finally, we show that the semantics of all typable terms, at any order and in the presence of iteration, has a regular-language representation and thus the restricted observational equivalence is decidable.

1 Introduction

Game semantics has emerged as a powerful paradigm for giving semantics to a spectrum of programming languages ranging from purely functional languages to those with non-functional features such as control operators and references [1–6]. Recently it has been developing in a new, algorithmic direction. Hankin and Malacaria have applied it to program analysis [7,8]. Ghica and McCusker [9] found that the game semantics of a second-order fragment of a procedural language can be captured by regular languages, demonstrating a new, semantics-directed, approach to software model-checking [10]. The approach has subsequently been extended in various directions: to third order [11], call-by-value [12,13], Hoare-style assertions [14] and specification [15].

In this paper we propose a game-based framework for compositional model checking of concurrent programs. Although a fully abstract game model for a

⋆ Work funded by British EPSRC, Canadian NSERC and St John's College, Oxford.

concurrent programming language exists [16], it seems unsuitable as a model of computation for model-checking applications. We can show that observational equivalence, even at second order in the absence of recursion, is not decidable. The sources of non-finitary behaviour are the free identifiers of first or higher-order types, which correspond to procedures using an argument in an unbounded number of concurrent threads of computation.

In the game model, active threads at any moment correspond to *pending questions* in a play. Hence, we constrain plays by placing bounds on the allowable number of pending questions and enforce these restrictions syntactically using a type system augmented with resource bounds. The key differences between this type system and the standard type system, are the "linearization" of application and parallel composition, i.e. requiring the environments of the two sub-terms to be disjoint. We also revise the *contraction rule* to count the number of contracted occurrences of a variable. We call this type system *Syntactic Control of Concurrency* (SCC); it is a generalization of *Serially Reentrant Algol* (SRA), a type system introduced by Abramsky to identify higher-order terms of a sequential language denotable by "pointer-free" finitary strategies [17].

The bounds imposed on the number of pending questions by SCC can be seen as a kind of *assume-guarantee* reasoning (see e.g. [18]): bounds on the behaviour of the *Opponent* represent *assumptions* on the behaviour of the environment, while bounds on the behaviour of the *Proponent* represent *guarantees* on the behaviour of the system. Typability can be seen as composition, made possible by the fact that the guarantees and the assumptions match. Unsurprisingly, not all terms of the original language admit a resource-bounding typing.

Resource-sensitive type systems are an area of research with numerous applications; the examples mentioned below are only entry points to a vast literature. The nature of the controlled resource is usually duration [19] or space [20]; applications of such systems are as diverse as execution in embedded systems [21], memory management [22], compilation to hardware [23] or proof-carrying code [24]. Type systems have been also used to control more abstract resources, such as variable usage for improved compilation [25] or interference effects for specification and verification [26].

The motivation behind SCC is to isolate (open) terms with finitary models for the purpose of automated verification. The notion of resource in SCC, which we may call *active threads of computation*, has a computational meaning, but it is primarily motivated by the game-semantic analysis of the language [16]. The main thrust of the paper is thus semantic; we plan to investigate the type-theoretic issues of SCC separately.

$$\Gamma, x : \theta \vdash x : \theta \qquad \Gamma \vdash \mathbf{skip} : \mathbf{com} \qquad \frac{m \in \{\, 0, \cdots, \mathrm{MAX} \,\}}{\Gamma \vdash m : \mathbf{exp}}$$

$$\frac{\Gamma \vdash M : \mathbf{com} \qquad \Gamma \vdash N : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash M; N : \mathbf{com}, \mathbf{exp}} \qquad \frac{\Gamma \vdash M : \mathbf{com} \qquad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash M \,||\, N : \mathbf{com}}$$

$$\frac{\Gamma \vdash M : \mathbf{var} \qquad \Gamma \vdash N : \mathbf{exp}}{\Gamma \vdash M := N : \mathbf{com}} \qquad \frac{\Gamma \vdash M : \mathbf{var}}{\Gamma \vdash !M : \mathbf{exp}}$$

$$\frac{\Gamma, x : \theta \vdash M : \theta'}{\Gamma \vdash \lambda x.M : \theta \to \theta'} \qquad \frac{\Gamma \vdash M : \theta \to \theta' \qquad \Gamma \vdash N : \theta}{\Gamma \vdash MN : \theta'} \qquad \frac{\Gamma \vdash M : \theta \to \theta}{\Gamma \vdash \mathbf{fix}(M) : \theta}$$

$$\frac{\Gamma \vdash M : \mathbf{exp} \qquad \Gamma \vdash N_1, N_2 : \beta}{\Gamma \vdash \mathbf{if}\, M\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2 : \beta} \qquad \frac{\Gamma \vdash M : \mathbf{exp} \qquad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash \mathbf{while}\, M\, \mathbf{do}\, N : \mathbf{com}}$$

$$\frac{\Gamma \vdash M : \mathbf{sem}}{\Gamma \vdash \mathbf{release}(M) : \mathbf{com}} \qquad \frac{\Gamma \vdash M : \mathbf{sem}}{\Gamma \vdash \mathbf{grab}(M) : \mathbf{com}}$$

$$\frac{\Gamma, x : \mathbf{var} \vdash M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash \mathbf{newvar}\, x := m\, \mathbf{in}\, M : \mathbf{com}, \mathbf{exp}}$$

$$\frac{\Gamma, x : \mathbf{sem} \vdash M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash \mathbf{newsem}\, x := m\, \mathbf{in}\, M : \mathbf{com}, \mathbf{exp}}$$

$$\frac{\Gamma \vdash M : \mathbf{com} \qquad \Gamma \vdash N : \mathbf{com}}{\Gamma \vdash \mathbf{mksem}\, MN : \mathbf{sem}} \qquad \frac{\Gamma \vdash M : \mathbf{exp} \to \mathbf{com} \qquad \Gamma \vdash N : \mathbf{exp}}{\Gamma \vdash \mathbf{mkvar}\, MN : \mathbf{var}}$$

Fig. 1. ICA typing rules

2 ICA and Its Game Model

Our object language, Idealized Concurrent Algol (ICA), is Idealized Algol over the finite data-type $\{\, 0, \ldots, \mathrm{MAX} \,\}$ ($\mathrm{MAX} > 0$) extended with parallel composition ($||$) and binary semaphores. Its types are generated by the grammar given below

$$\beta ::= \mathbf{com} \mid \mathbf{exp} \mid \mathbf{var} \mid \mathbf{sem} \qquad \theta ::= \beta \mid \theta \to \theta$$

and the typing judgements are displayed in Figure 1. Semaphores can be manipulated using two (blocking) primitives, $\mathbf{grab}(S)$ and $\mathbf{release}(S)$, which grab and respectively release the semaphore. We also use variable and semaphore constructors **mkvar** and **mksem** (this is necessary for the full abstraction results: Theorems 5 and 19; **mkvar** was first introduced for this purpose in [3]).

The operational semantics is defined using a (small-step) transition relation $\Sigma \vdash M,\, s \longrightarrow M',\, s'$. Σ is a set of names of variables denoting *memory cells* and those of semaphores denoting *locks*; s, s' are states, i.e. functions $s, s' : \Sigma \to \{\, 0, \cdots, \mathrm{MAX} \,\}$, and M, M' are terms. The basic reduction rules are given in Figure 2, where c stands for any language constant (m or **skip**).

$$
\begin{aligned}
\Sigma \vdash \mathbf{skip} \,||\, \mathbf{skip},\ s &\longrightarrow \mathbf{skip},\ s \\
\Sigma \vdash \mathbf{skip}; c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \mathbf{newvar}\, x := m\, \mathbf{in}\, c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \mathbf{newsem}\, x := m\, \mathbf{in}\, c,\ s &\longrightarrow c,\ s \\
\Sigma \vdash \mathbf{if}\, 0\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2,\ s &\longrightarrow N_2,\ s \\
\Sigma \vdash \mathbf{if}\, m\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2,\ s &\longrightarrow N_1,\ s, \quad m \neq 0 \\
\Sigma \vdash\, !v,\ s \otimes (v \mapsto m) &\longrightarrow m,\ s \otimes (v \mapsto m) \\
\Sigma \vdash v := m',\ s \otimes (v \mapsto m) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto m') \\
\Sigma \vdash \mathbf{grab}(v),\ s \otimes (v \mapsto 0) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto 1) \\
\Sigma \vdash \mathbf{release}(v),\ s \otimes (v \mapsto m) &\longrightarrow \mathbf{skip},\ s \otimes (v \mapsto 0), \quad m \neq 0 \\
\Sigma \vdash (\lambda x.M)N,\ s &\longrightarrow M[N/x],\ s \\
\Sigma \vdash \mathbf{fix}\, M,\ s &\longrightarrow M(\mathbf{fix}\, M),\ s \\
\Sigma \vdash (\mathbf{mkvar}\, MN) := M',\ s &\longrightarrow MM',\ s \\
\Sigma \vdash\, !(\mathbf{mkvar}\, MN),\ s &\longrightarrow N,\ s \\
\Sigma \vdash \mathbf{grab}(\mathbf{mksem}\, MN),\ s &\longrightarrow M,\ s \\
\Sigma \vdash \mathbf{release}(\mathbf{mksem}\, MN),\ s &\longrightarrow N,\ s
\end{aligned}
$$

Fig. 2. Reduction rules for ICA

In-context reduction is given by the schemata:

$$
\frac{\Sigma, v \vdash M[v/x], s \otimes (v \mapsto m) \longrightarrow M', s' \otimes (v \mapsto m') \quad M \neq c}{\Sigma \vdash \mathbf{newvar}\, x := m\, \mathbf{in}\, M, s \longrightarrow \mathbf{newvar}\, x := m'\, \mathbf{in}\, M'[x/v], s'}
$$

$$
\frac{\Sigma, v \vdash M[v/x], s \otimes (v \mapsto m) \longrightarrow M', s' \otimes (v \mapsto m') \quad M \neq c}{\Sigma \vdash \mathbf{newsem}\, x := m\, \mathbf{in}\, M, s \longrightarrow \mathbf{newsem}\, x := m'\, \mathbf{in}\, M'[x/v], s'}
$$

$$
\frac{\Sigma \vdash M,\ s \longrightarrow M',\ s'}{\Sigma \vdash \mathcal{E}[M],\ s \longrightarrow \mathcal{E}[M'],\ s'}
$$

where reduction contexts $\mathcal{E}[-]$ are produced by the grammar:

$$
\begin{aligned}
\mathcal{E}[-] ::= {} & [-] \mid \mathcal{E}; N \mid (\mathcal{E} \,||\, N) \mid (M \,||\, \mathcal{E}) \mid \mathcal{E}N \\
& \mid \mathbf{if}\, \mathcal{E}\, \mathbf{then}\, N_1\, \mathbf{else}\, N_2 \mid\, !\mathcal{E} \mid \mathcal{E} := m \mid M := \mathcal{E} \mid \mathbf{grab}(\mathcal{E}) \mid \mathbf{release}(\mathcal{E}).
\end{aligned}
$$

We consider an *angelic* notion of termination: we say that a term M terminates in state s, written $M, s \Downarrow$, if there exists a terminating evaluation at start state s: $\exists s',\ M, s \longrightarrow^* c, s'$, with $c \in \{\, 0, \cdots, \mathrm{MAX} \,\}$ or $c = \mathbf{skip}$. If M is closed and $M, \emptyset \Downarrow$ we write $M \Downarrow$. We define the *observational approximation* relation contextually: $\Gamma \vdash M_1 \sqsubseteq_{\approx} M_2$ holds if and only if $\forall \mathcal{C}[-] : \mathbf{com}$, $\mathcal{C}[M_1] \Downarrow$ implies $\mathcal{C}[M_2] \Downarrow$, where $\mathcal{C}[M_i]$ are closed programs of type $\mathbf{com}$. *Observational may-equivalence* ($\Gamma \vdash M_1 \cong M_2$) is then defined as $\Gamma \vdash M_1 \sqsubseteq_{\approx} M_2$ and $\Gamma \vdash M_2 \sqsubseteq_{\approx} M_1$.

In [16] we have given a game model which is fully abstract for $\underset{\sim}{\sqsubset}$ and $\cong$. We give a sketch of the model.

Definition 1 *An* arena A *is a triple* $\langle M_A, \lambda_A, \vdash_A \rangle$ *where:*

- M_A *is a set of* moves*;*
- $\lambda_A : M_A \to \{ O, P \} \times \{ Q, A \}$ *is a function determining for each* $m \in M_A$ *whether it is an* Opponent *or a* Proponent move*, and a* question *or an* answer*; we write* $\lambda_A^{OP}, \lambda_A^{QA}$ *for the composite of* λ_A *with respectively the first and second projections;*
- $\vdash_A$ *is a binary relation on* M_A*, called* enabling*, satisfying: if* $m \vdash_A n$ *for no* m *then* $\lambda_A(n) = (O, Q)$*, if* $m \vdash_A n$ *then* $\lambda_A^{OP}(m) \neq \lambda_A^{OP}(n)$*, and if* $m \vdash_A n$ *then* $\lambda_A^{QA}(m) = Q$*.*

If $m \vdash_A n$ we say that m *enables* n. We shall write I_A for the set of all moves of A which have no enabler; such moves are called *initial*. Note that an initial move must be an Opponent question.

The *product* $(A \times B)$ and *arrow* $(A \Rightarrow B)$ arenas are defined by:

$$\begin{array}{ll} M_{A \times B} = M_A + M_B & M_{A \Rightarrow B} = M_A + M_B \\ \lambda_{A \times B} = [\lambda_A, \lambda_B] & \lambda_{A \Rightarrow B} = [\langle \lambda_A^{PO}, \lambda_A^{QA} \rangle, \lambda_B] \\ \vdash_{A \times B} = \vdash_A + \vdash_B & \vdash_{A \Rightarrow B} = \vdash_A + \vdash_B + \{ (b, a) \mid b \in I_B \text{ and } a \in I_A \} \end{array}$$

where $\lambda_A^{PO}(m) = O$ iff $\lambda_A^{OP}(m) = P$.

In arenas used to interpret base types all questions are initial and P-moves answer them as detailed in the table below, where $m \in \{ 0, \cdots, \mathrm{MAX} \}$.

Arena	O-question	P-answers	Arena	O-question	P-answers
$[\![\mathbf{com}]\!]$	*run*	*ok*	$[\![\mathbf{exp}]\!]$	q	m
$[\![\mathbf{var}]\!]$	*read*	m	$[\![\mathbf{sem}]\!]$	*grab*	*ok*
	write(m)	*ok*		*release*	*ok*

A *justified sequence* in arena A is a finite sequence of moves of A equipped with pointers. The first move is initial and has no pointer, but each subsequent move n must have a unique pointer to an earlier occurrence of a move m such that $m \vdash_A n$. We say that n is (explicitly) justified by m or, when n is an answer, that n answers m. Note that interleavings of several justified sequences may not be justified sequences; instead we shall call them *shuffled sequences*. If a question does not have an answer in a justified sequence, we say that it is *pending* (or *open*) in that sequence. In what follows we use the letters q and

a to refer to question- and answer-moves respectively, m denotes arbitrary moves. Not all justified sequences are valid. In order to constitute a legal play, a justified sequence must satisfy a well-formedness condition which reflects the "static" style of concurrency of our programming language: any process starting sub-processes must wait for the children to terminate in order to continue. In game terms, if a question is answered then that question and all questions justified by must have been answered (exactly once). This is spelled out as follows:

Definition 2 *The set P_A of* positions *(or* plays*) over A consists of the justified sequences s over A which satisfy the two conditions below.*

FORK *: In any prefix $s' = \cdots\ q \overset{\frown}{\cdots\ m}$ of s, the question q must be pending before m is played.*

WAIT *: In any prefix $s' = \cdots\ q \overset{\frown}{\cdots\ a}$ of s, all questions justified by q must be answered.*

For two shuffled sequences s_1 and s_2, $s_1 \amalg s_2$ denotes the set of all interleavings of s_1 and s_2. For two sets of shuffled sequences S_1 and S_2, $S_1 \amalg S_2 = \bigcup_{s_1 \in S_1, s_2 \in S_2} s_1 \amalg s_2$. Given a set X of shuffled sequences, we define $X^0 = X$, $X^{i+1} = X^i \amalg X$. Then $X^{\circledast}$, called *iterated shuffle* of X, is defined to be $\bigcup_{i \in \mathbb{N}} X^i$.

We say that a subset σ of P_A is *O-complete* if $s \in \sigma$ and $so \in P_A$, where o is an occurrence of an O-move, entail $so \in \sigma$.

Definition 3 *A* strategy *σ on A (written $\sigma : A$) is a prefix-closed subset of P_A, which is* O-complete*.*

Strategies $\sigma : A \Rightarrow B$ and $\tau : B \Rightarrow C$ are composed in the standard way, by considering all possible interactions of positions from τ with shuffled sequences of $\sigma^{\circledast}$ in the shared arena B, then hiding the B moves.

The model consists of *saturated* strategies only: the saturation condition stipulates that all possible (sequential) observations of (parallel) interactions must be present in a strategy: actions of the environment can always be observed earlier if possible, actions of the program can always be observed later. To formalize this, for any arena A a preorder $\preceq$ on P_A is defined, as the least transitive relation $\preceq$ satisfying $s_0 \cdot o \cdot s_1 \cdot s_2 \preceq s_0 \cdot s_1 \cdot o \cdot s_2$ and $s_0 \cdot s_1 \cdot p \cdot s_2 \preceq s_0 \cdot p \cdot s_1 \cdot s_2$ for all s_0, s_1, s_2 where o is an O-move and p is a P-move. In the above pairs of positions moves on the left-hand-side of $\preceq$ have the same justifiers as on the right-hand-side.

Definition 4 *A strategy σ is* saturated *iff $s \in \sigma$ and $s' \preceq s$ imply $s' \in \sigma$.*

The two saturation conditions, in various formulations, have a long pedigree in the semantics of concurrency. For example, they have been used by Udding

to describe propagation of signals across wires in delay-insensitive circuits [27] and by Josephs *et al* to specify the relationship between input and output in asynchronous systems with channels [28]. Laird has been the first to adopt them in game semantics, in his model of Idealized CSP [29].

Arenas and saturated strategies form a Cartesian closed category $\mathcal{G}_{\mathrm{sat}}$ in which $\mathcal{G}_{\mathrm{sat}}(A, B)$ consists of saturated strategies on $A \Rightarrow B$. The identity strategy is defined by "saturating" the alternating positions $s \in P_{A_1 \Rightarrow A_2}$ such that $\forall\, t \sqsubseteq_{\mathrm{even}} s,\ t \upharpoonright A_1 = t \upharpoonright A_2$, which gives rise to the behaviour of an unbounded buffer (we use A_1 and A_2 to distinguish the two copies of A in the arena $A \Rightarrow A$).

Other elements of the syntax are interpreted by the least saturated strategies generated by the plays from the table below:

;	$q_1\, run\, ok\, q_0\, a_0\, a_1$	$\|\|$	$run_2\, run_0\, run_1\, ok_0\, ok_1\, ok_2$
:=	$run_2\, q_1\, m_1\, write(m)_0\, ok_0\, ok_2$	!	$q\, read\, m\, m$
grab	$run_1\, grab_0\, ok_0\, ok_1$	**release**	$run_1\, release_0\, ok_0\, ok_1$

newvar $x := m$	$q\, q\, (read\, m)^*\, (\sum_{i=0}^{\mathrm{MAX}} (write(i)\, ok\, (read\, i)^*))^*\, a\, a$
newsem $x := 0$	$q\, q\, (grab\, ok\, release\, ok)^*\, (grab\, ok + \epsilon)\, a\, a$
newsem $x := 1$	$q\, q\, (release\, ok\, grab\, ok)^*\, (release\, ok + \epsilon)\, a\, a.$

Here we follow a convention (see e.g. [9]) that uses subscripts to distinguish copies of the same move.

As shown in [16], $\mathcal{G}_{\mathrm{sat}}$ is fully abstract for $\cong$ in the sense mentioned below. $\mathsf{comp}(\sigma)$ denotes the set of non-empty *complete* plays of a strategy σ, i.e. those in which all questions have been answered.

Theorem 5 $\Gamma \vdash M_1 \sqsubseteq\!\!\!\!\sim\, M_2 \iff \mathit{comp}([\![\Gamma \vdash M_1]\!]) \subseteq \mathit{comp}([\![\Gamma \vdash M_2]\!])$. *Hence,* $\Gamma \vdash M_1 \cong M_2 \iff \mathit{comp}([\![\Gamma \vdash M_1]\!]) = \mathit{comp}([\![\Gamma \vdash M_2]\!])$.

3 Undecidability of ICA May-Equivalence

A *Minsky machine* [30] is a state machine with two unbounded counters c_1, c_2. Formally, it can be viewed as a tuple $\langle Q, q_0, F, \delta \rangle$, where Q is the set of states partitioned into disjoint subsets Q_1^{INC}, Q_2^{INC}, Q_1^{DEC}, Q_2^{DEC} with a designated set of final states F ($q_0 \notin F$) and where δ denotes the two groups of functions: $\delta_1 : Q_1^{\mathrm{INC}} \to Q$, $\delta_2 : Q_2^{\mathrm{INC}} \to Q$ and $\delta_1^0 : Q_1^{\mathrm{DEC}} \to Q$, $\delta_2^0 : Q_2^{\mathrm{DEC}} \to Q$, $\delta_1^+ : Q_1^{\mathrm{DEC}} \to Q$, $\delta_2^+ : Q_1^{\mathrm{DEC}} \to Q$. The machine starts from an initial state q_0. The initial values of both counters are 0. When the machine is in state

$q \in Q_i^{\mathrm{INC}}$, the counter c_i is incremented by 1 and the machine moves to $\delta_i(q)$. When in state $q \in Q_i^{\mathrm{DEC}}$, the next step depends on whether the value of c_i is zero. If so, the machine enters $\delta_i^0(q)$. Otherwise c_i is decremented by 1 and the machine moves to $\delta_i^+(q)$. We say that a Minsky machine machine *halts* if a final state is entered and the values of both counters are then 0. It is well-known [30] that the halting problem for Minsky machines is not decidable and we use this to show that neither is observational equivalence.

Theorem 6 *ICA may-equivalence is undecidable.*

PROOF. Let $\Omega_{\mathbf{com}}$ stand for $\mathbf{fix}(\lambda x.x) : \mathbf{com}$, i.e. $\Omega_{\mathbf{com}}$ is the divergent command. We show how, given a Minsky machine, one can define a term $g : \mathbf{com} \to \mathbf{com} \vdash M : \mathbf{com}$ such that $g : \mathbf{com} \to \mathbf{com} \vdash M \cong \Omega_{\mathbf{com}}$ if and only if the associated machine does not halt. By Theorem 5 it suffices to show that $\mathsf{comp}([\![g : \mathbf{com} \to \mathbf{com} \vdash M : \mathbf{com}]\!])$ is not empty iff the machine halts.

The construction of M takes advantage of the fact that the free identifier $g : \mathbf{com} \to \mathbf{com}$ represents an indeterminate procedure which can use its argument in a variety of ways (possibly in parallel; the uses may be interleaved or overlapping). This intuition is captured by the use of iterated shuffle in the game model. Using semaphores we are going to restrict the shape of the interleavings (possibly) generated by g in such a way that terminating computations (complete positions) can only arise from a halting run of the Minsky machine.

Let us write $[B]$ for $\mathbf{if}\, B\, \mathbf{then\, skip\, else}\, \Omega_{\mathbf{com}}$. For $i = 1, 2$ we define the following terms

$$\begin{aligned} I_i &\equiv \mathbf{grab}(S); [!ST \in Q_i^{\mathrm{INC}}]; ST := \delta_i(!ST); \mathbf{release}(S) \\ D_i^+ &\equiv \mathbf{grab}(S); [!ST \in Q_i^{\mathrm{DEC}}]; ST := \delta_i^+(!ST); \mathbf{release}(S) \\ D_i^0 &\equiv \mathbf{grab}(S); [!ST \in Q_i^{\mathrm{DEC}}]; ST := \delta_i^0(!ST); \mathbf{release}(S), \end{aligned}$$

which will correspond to the three kinds of actions on the counter c_i. Note that each of the above terms is protected with the same semaphore, so terminating computations can only emerge from interleavings of the six kinds of terms where they are processed in a sequence. Hence, to complete the argument, it suffices to impose additional restrictions guaranteeing that only the sequence simulating actions of the Minsky machine leads to convergence.

We first focus on ensuring that zero tests (modelled by D_i^+ or D_i^0) are handled correctly. By a *simple* history of a counter we mean a (possibly empty) series of increments and decrements starting at value 0 resulting in value 0, possibly followed by a zero test. Observe that the full history of a counter in a halting Minsky machine is a sequence of simple histories. Then the term

$$C_i \equiv \mathbf{grab}(S_i); g(I_i; D_i^+); (D_i^0\ \mathbf{or\ skip}); \mathbf{release}(S_i)$$

corresponds to all simple histories of c_i: because copies of I_i and D_i^+ will never overlap in terminating computations, $g(I_i; D_i^+)$ corresponds to all potential sequences of increments and decrements of c_i leading from value 0 to 0. Note also the use of semaphores S_i which will guarantee that different copies of C_1 (C_2 respectively), i.e. simple histories of each counter, can be interleaved only sequentially, while the sequence of C_1's and the sequence of C_2's can interact freely. This ensures that $g(C_1 \,\mathbf{or}\, C_2)$, where

$$C_1 \,\mathbf{or}\, C_2 \equiv \mathbf{newvar}\, x \,\mathbf{in}\, (x := 0 \,||\, x := 1); \mathbf{if}\, !x \,\mathbf{then}\, C_1 \,\mathbf{else}\, C_2,$$

will represent all sequences of $I_1, I_2, D_1^+, D_1^0, D_2^+, D_2^0$ in which before each D_i^+ the number of I_i's always exceeds that of D_i^+'s and before each D_i^0 we have an equal number of I_i's and D_i^+'s.

Finally, in order to capture a halting run of the given Minsky machine, we have to make sure that the sequences above are consistent with state changes of the machine. This is however already guaranteed by assertions of the form $[!ST \in \cdots]$ in I_i, D_i^+, D_i^0 whose presence we have ignored in our argument so far. M can thus be taken to be:

$$\begin{array}{lll} g : \mathbf{com} \to \mathbf{com} & \vdash & \mathbf{newvar}\ ST := q_0\ \mathbf{in} \\ & & \mathbf{newsem}\ S, S_1, S_2 := 0, 0, 0\ \mathbf{in}\ g(C_1 \,\mathbf{or}\, C_2); [!ST \in F]. \end{array}$$

□

4 SCC: a Resource-Bounding Type System

The simulation above is possible because free identifiers $\mathbf{com} \to \mathbf{com}$ correspond to functions that investigate the argument an arbitrary number of times (possibly in parallel). Therefore the key to regaining decidability is to restrict the number of times an argument is used concurrently. However, we need not restrict the number of sequential uses, to allow for iteration and all sorts of interesting procedural programs.

The type system is for the recursion-free fragment with **while**. Divergence, $\Omega_{\mathbf{com}}$, can then be defined by $\mathbf{while}\, 1 \,\mathbf{do}\, \mathbf{skip}$. Types are generated by the following grammar:

$$\beta ::= \mathbf{com} \mid \mathbf{exp} \mid \mathbf{var} \mid \mathbf{sem} \qquad \theta ::= \beta \mid \gamma \to \theta \qquad \gamma ::= \theta^n.$$

The numbers that label the left-hand side of a function type will be called *resource bounds*. An occurrence m of a resource bound in a type θ is an *assume* (respectively *guarantee*) if it occurs in the left-hand scope of an even

(respectively odd) number of $\rightarrow$'s in θ. Formally, m is an assume (a guarantee) in θ iff $\theta = \mathcal{A}[m]$ ($\theta = \mathcal{G}[m]$):

$$\mathcal{G}[\ ] ::= \theta_1^{[\ ]} \rightarrow \theta_2 \mid \theta^n \rightarrow \mathcal{G}[\ ] \mid \mathcal{A}[\ ]^n \rightarrow \theta,$$

$$\mathcal{A}[\ ] ::= \theta^n \rightarrow \mathcal{A}[\ ] \mid \mathcal{G}[\ ]^n \rightarrow \theta.$$

For instance, 3 in $(\mathbf{com}^3 \rightarrow \mathbf{com})^4 \rightarrow \mathbf{com}$ is an assume and 4 is a guarantee. Assumes and guarantees will turn out to correspond to the Opponent/Player polarity in game semantics.

Assumes characterize the behaviour of the program context and guarantees characterize that of the program. The assumes of a typing judgement $\theta_1^{n_1}, \ldots, \theta_k^{n_k} \vdash M : \theta$ are the assumes in θ along with the guarantees in $\theta_1, \ldots, \theta_k$. The guarantees of a typing judgement are the guarantees of θ, the assumes in $\theta_1, \ldots, \theta_k$ and $n_1, \ldots, n_k$.

We use types of this form to approximate the maximum number of concurrent sub-threads of computation at any moment. This estimate is subject to assumes on the environment. Intuitively, if a program has a type θ, then provided the environment behaves according to the assumes, the program's behaviour satisfies the guarantees. In this spirit we introduce a sub-typing relation which can be taken to correspond to weakening the constraints imposed by SCC.

$$\beta \leq \beta \qquad \frac{n_1 \leq n_2}{\theta^{n_2} \leq \theta^{n_1}} \qquad \frac{\gamma_2 \leq \gamma_1 \quad \theta_1 \leq \theta_2}{\gamma_1 \rightarrow \theta_1 \leq \gamma_2 \rightarrow \theta_2}.$$

Intuitively, a subtype gives a less precise approximation: higher on the behaviour of the program and lower for the environment. In the latter case, the bound is considered inferior because it applies to a weaker behaviour of the environment.

The SCC typing rules are given in Figure 3. Typing judgements are of the form $\Gamma \vdash_r M : \theta$, where $\Gamma = x_1{:}\theta_1^{n_1}, \ldots, x_k{:}\theta_k^{n_k}$; we write $n\Gamma = x_1{:}\theta_1^{n \cdot n_1}, \ldots, x_k{:}\theta_k^{n \cdot n_k}$. Note that the typing rules make a distinction between parallel and sequential composition. Parallel composition and application have multiplicative rules, in which the contexts are required to be disjoint, as opposed to the rules for sequential operators ($\Box$ can stand for $;$, $:=$, $!$, **grab**, **release**) including branching and iteration. In order to be able to use identifiers (e.g. semaphores) in concurrent threads, a *contraction* rule is necessary; we modify it so that the guaranteed bounds on the contracted variables are accumulated into the new variable.

Remark 7 *The rule for application is also multiplicative. The reason is that call-by-name application is a peculiar form of concurrency in which the computation carried out by the function is interleaved with that of its argument, albeit in a highly constrained fashioned. For instance, if F is a first-order*

Axioms:

$$x : \theta^1 \vdash_r x : \theta \qquad \vdash_r \mathbf{skip} : \mathbf{com} \qquad \frac{m \in \{\, 0, \cdots, \mathrm{MAX} \,\}}{\vdash_r m : \mathbf{exp}}$$

Additive rules:

$$\frac{\Gamma \vdash_r M : \theta}{\Gamma, x : \gamma \vdash_r M : \theta} \qquad \frac{\Gamma \vdash_r M : \theta \qquad \theta \leq \theta'}{\Gamma \vdash_r M : \theta'}$$

$$\frac{\{\Gamma \vdash_r M : \theta_1\} \qquad \Gamma \vdash_r N : \theta_2}{\Gamma \vdash_r \{M\} \,\square\, N : \theta_3} \qquad \frac{\Gamma, x : \gamma \vdash_r M : \theta}{\Gamma \vdash_r \lambda x.M : \gamma \to \theta}$$

$$\frac{\Gamma \vdash_r M : \mathbf{exp} \qquad \Gamma \vdash_r N_1, N_2 : \beta}{\Gamma \vdash_r \mathbf{if}\, M \,\mathbf{then}\, N_1 \,\mathbf{else}\, N_2 : \beta} \qquad \frac{\Gamma \vdash_r M : \mathbf{exp} \qquad \Gamma \vdash_r N : \mathbf{com}}{\Gamma \vdash_r \mathbf{while}\, M \,\mathbf{do}\, N : \mathbf{com}}$$

$$\frac{\Gamma, x : \mathbf{var}^n \vdash_r M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash_r \mathbf{newvar}\, x := m \,\mathbf{in}\, M : \mathbf{com}, \mathbf{exp}}$$

$$\frac{\Gamma, x : \mathbf{sem}^n \vdash_r M : \mathbf{com}, \mathbf{exp}}{\Gamma \vdash_r \mathbf{newsem}\, x := m \,\mathbf{in}\, M : \mathbf{com}, \mathbf{exp}}$$

$$\frac{\Gamma \vdash_r M : \mathbf{com} \qquad \Gamma \vdash_r N : \mathbf{com}}{\Gamma \vdash_r \mathbf{mksem}\, MN : \mathbf{sem}}$$

$$\frac{\Gamma \vdash_r M : \mathbf{exp}^n \to \mathbf{com} \qquad \Gamma \vdash_r N : \mathbf{exp}}{\Gamma \vdash_r \mathbf{mkvar}\, MN : \mathbf{var}}$$

Multiplicative rules:

$$\frac{\Gamma, x : \theta^{n_1}, y : \theta^{n_2} \vdash_r M : \theta'}{\Gamma, x : \theta^{n_1+n_2} \vdash_r M[x/y] : \theta'} \qquad \frac{\Gamma \vdash_r M : \mathbf{com} \qquad \Delta \vdash_r N : \mathbf{com}}{\Gamma, \Delta \vdash_r M \,||\, N : \mathbf{com}}$$

$$\frac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n\Delta \vdash_r MN : \theta'}$$

Fig. 3. SCC typing rules

function, any computation arising in an application $F(M)$ also arises in the parallel composition $\cdots F(\cdots) \cdots \,||\, \cdots M \cdots \,||\, \cdots \,||\, \cdots M \cdots$, where the ellipses stand for code manipulating semaphores so that the right interleaving of effects is enforced [16]. A multiplicative application rule is also used in SRA [17].

Example 8 *For any $n \in \mathbb{N}$ we have*

(1) $\vdash_r \lambda f x.f(f(x)) : (\mathbf{com}^n \to \mathbf{com})^{n+1} \to (\mathbf{com}^{n^2} \to \mathbf{com})$
(2) $\vdash_r \lambda f x.f(x); f(x) : (\mathbf{com}^n \to \mathbf{com})^1 \to (\mathbf{com}^n \to \mathbf{com})$
(3) $\vdash_r \lambda f x.f(x) \,||\, f(x) : (\mathbf{com}^n \to \mathbf{com})^2 \to (\mathbf{com}^{2n} \to \mathbf{com})$
(4) $\vdash_r \lambda f.f(f\mathbf{skip}) : (\mathbf{com}^n \to \mathbf{com})^{n+1} \to \mathbf{com}$

(5) $\vdash_r \lambda g.g(\lambda x.g(\lambda y.x)) : ((\mathbf{com}^n \to \mathbf{com})^n \to \mathbf{com})^{n+1} \to \mathbf{com}$

SCC enjoys the standard syntactic properties of a typed lambda calculus (basis, generation, sub-term, substitution and subject reduction lemmas) [31]. We also have the easily derivable dual of the subsumption law:

$$\frac{\Gamma, x : \theta' \vdash_r M : \theta'' \qquad \theta \leq \theta'}{\Gamma, x : \theta \vdash_r M : \theta''} \ .$$

Not all ICA terms are typable in SCC. However, if one attempts to prove typability by induction on the structure of ICA derivations, it turns out that only the application rule does not preserve it. This is because the corresponding SCC rule requires that the bounds of the argument match those of the function term. For example, the application of the term 5 to term 4 above is untypable.

Given the bounds for environment, SCC can be used to certify bounds for the program.

Definition 9 *An ICA term $\Gamma \vdash M : \theta$ is r-typable if for any assignment of assumes there exists an assignment of guarantees such that when we adorn Γ, θ with these bounds we get Γ', θ' such that $\Gamma' \vdash_r M : \theta'$. We shall write η^{a}, η^{g} respectively for the two assignments.*

Since not all terms are typable, not all terms are r-typable. However, there is a wide class of r-typable (and so typable) terms. The key to regaining typability lies in restricting the shape of possible applications. The Lemma below exhibits two instances where typability is preserved (a combination of the two is also possible).

Lemma 10 *(1) Any β-normal ICA term is r-typable.*
(2) Any ICA term in which function arguments are of first order or base type is r-typable.

PROOF. We reason by induction on the ICA typing rules. All cases except the application rule are routine appeals to the induction hypothesis. For rules with more than one premise it is necessary to apply the dual law given above to find a common typing (by using the higher of the guarantees provided by the several appeals to the induction hypothesis). Additionally, for $||$, contraction has to be used. Finally, we consider the restricted forms of application.

(1) If a term is β-normal then all applications have the form $\Gamma \vdash f M_1 \cdots M_k$. For simplicity, we will assume that $k = 1$ (the argument for $k > 1$ follows the same pattern).
Suppose $\Gamma \vdash f M_1 : \theta_2$ and η^{a} is an assignment of assumes to the typing

judgement. Let $\eta^{\mathrm{a}}_r = \eta^{\mathrm{a}} \upharpoonright \theta_2$. Note that $f : \theta_1 \to \theta_2$, for some θ_1, must be present in Γ, so η^{a} also defines bounds for the associated occurrence of $\theta_1 \to \theta_2$. Let $\eta^{\mathrm{a}}_f = \eta^{\mathrm{a}} \upharpoonright (\theta_1 \to \theta_2)$.

Consider $\Gamma \vdash M_1 : \theta_1$ and the assignment of assumes in which the bounds for Γ are the same as in η^{a} and those for θ_1 are determined by η^{a}_f. By IH we get $\Gamma' \vdash_r M_1 : \theta'_1$. Define the resource type θ'_2 by decorating θ_2 with assumes given by η^{a}_r and guarantees which are assumes in η^{a}_f. Then we have $f : (\theta'_1 \to \theta'_2)^1 \vdash_r f : (\theta'_1 \to \theta'_2)$ and, consequently, $f : (\theta'_1 \to \theta'_2), \Gamma' \vdash_r fM_1 : \theta'_2$. Recall that Γ' will contain an occurrence of f where the associated assumes are the same as those for f. Using the dual subsumption law we can make the guarantees match and finish by contracting f.

(2) Suppose $\Gamma \vdash MN : \theta$ and $\Gamma \vdash N : \beta \to \beta$. Let η^{a} be an assignment of assumes to the first judgement. By IH, using assumes from η^{a} for Γ, we have $\Gamma' \vdash_r N : \beta^{n'} \to \beta$. Now consider $\Gamma \vdash M : (\beta \to \beta) \to \theta$. By IH, using assumes from η^{a} for Γ and n' for the leftmost occurrence of β, we get $\Gamma'' \vdash_r M : (\beta^{n'} \to \beta)^n \to \theta'$. Hence, $\Gamma'', n\Gamma' \vdash_r MN : \theta'$. Because Γ'' and Γ' share the same assumes, we can unify the guarantees inside Γ' and Γ'' using the dual subsumption law and follow with contraction to get $\Gamma''' \vdash_r fM : \theta'$, where the assumes in Γ''' coincide with η^{a}. □

Using SCC we can define a new *observational approximation* relation $\sqsubseteq\!\!\!\approx_r$ using typable terms and contexts along with their bounds. Suppose $\Gamma \vdash_r M_1, M_2 : \theta$. In what follows we write $\vdash_r \mathcal{C}[M_i]$ to mean that $\mathcal{C}[M_i]$ is typable and its derivation is constructed using (possibly several copies of) the given derivation of the typing judgement $\Gamma \vdash_r M_i$, up to appropriate renaming of variables. We define $\Gamma \vdash_r M_1 \sqsubseteq\!\!\!\approx_r M_2$ to hold iff for all contexts $\mathcal{C}[-]$ such that $\vdash_r \mathcal{C}[M_i] : \mathbf{com}$ we have: $\mathcal{C}[M_1] \Downarrow$ implies $\mathcal{C}[M_2] \Downarrow$. Similarly, we write $\Gamma \vdash_r M_1 \cong_r M_2$ iff $\Gamma \vdash_r M_1 \sqsubseteq\!\!\!\approx_r M_2$ and $\Gamma \vdash_r M_2 \sqsubseteq\!\!\!\approx_r M_1$. In particular, the definition applies to the terms for which the above lemma holds. Note that no bound needs to be placed on the way M_i is used in $\mathcal{C}[M_i]$, the bounds concern only the way its free identifiers are trapped in context. In the definition of $\sqsubseteq\!\!\!\approx_r$ we require $\Gamma \vdash_r M_i : \theta$ to have the same annotations. If two terms are typable with the same assumed bounds, it is always possible to type them with the same guaranteed bounds by sub-typing.

Example 11 ([32]) *Consider the terms*

$$M_1 \equiv \mathbf{newvar}\, x := 0\, \mathbf{in}\, (p(x := !x + 1; x := !x + 1);\ \mathbf{if}\ even(!x)\ \mathbf{then}\ \Omega_{\mathbf{com}})$$
$$M_2 \equiv \mathbf{newvar}\, x := 0\, \mathbf{in}\, (p(x := !x + 2);\ \mathbf{if}\ even(!x)\ \mathbf{then}\ \Omega_{\mathbf{com}})$$

with $p : \mathbf{com} \to \mathbf{com}$. *Brookes has shown that in* sequential *Algol they are observationally equivalent, whereas in* concurrent *Algol they are not. In SCC we have* $p : (\mathbf{com}^1 \to \mathbf{com})^1 \vdash_r M_1 \cong_r M_2$*; but for any (assumed) bound*

$n > 1$, $p : (\mathbf{com}^n \to \mathbf{com})^1 \vdash_r M_1 \not\cong_r M_2$. *The reason is that the assumed bound of 1 only allows identifier p to be bound to a procedure which uses its argument sequentially. For example, context $\mathcal{C}[-] = (\lambda p.[-])(\lambda c.c \,||\, c)$ cannot trap $p : \mathbf{com}^1 \to \mathbf{com}$. On the other hand, context $\mathcal{C}[-] = (\lambda p.[-])(\lambda c.c; c)$ can trap $p : \mathbf{com}^n \to \mathbf{com}$ for any n.*

A formal proof of this example is immediate once the connection with game semantics is established in the following section.

5 The Game Model Revisited

We use the game model to interpret the annotations from the type system and to show how the model can be used to reason about $\sqsubseteq_{\approx r}, \cong_r$. In order to analyze the positions induced by terms in more detail we define a more refined games framework where plays can form a subset of P_A as opposed to the full P_A. In particular we are going to dissect the possibilities for the function space game $A \Rightarrow B$. To do that we introduce an auxiliary notion of games in which shuffled sequences are allowed (cf. [33]).

Definition 12 *A* bounded *game $\underline{A}$ is a pair $\langle A, R_A \rangle$ where A is an arena and R_A is a prefix-closed subset of $P_A^{\circledast}$.*

We also refer to the elements of R_A as plays and write $\mathsf{comp}(R_A)$ for the set of complete plays in R_A (those in which all questions are answered). The games of $\mathcal{G}_{\mathrm{sat}}$ can be viewed as bounded games where $R_A = P_A$.

Using bounded games we can define a more refined type hierarchy:

$$\begin{aligned}
\underline{A} \times \underline{B} &= (A \times B, R_A + R_B) \\
\underline{A} \otimes \underline{B} &= (A \times B, R_A \amalg R_B) \\
!\underline{A} &= (A, R_A^{\circledast}) \\
\underline{A} \multimap \underline{B} &= (A \Rightarrow B, \{\, s \in P_{A \Rightarrow B}^{\circledast} \mid s \upharpoonright A \in R_A,\ s \upharpoonright B \in R_B \,\}).
\end{aligned}$$

We can then construct an arrow type as

$$\underline{A} \Rightarrow \underline{B} = !\underline{A} \multimap \underline{B}.$$

We also have

$$!\underline{A} \otimes !\underline{B} = !(\underline{A} \times \underline{B}).$$

Note that where $R_A = P_A$, $R_B = P_B$ the $\times$ and $\Rightarrow$ constructions coincide with the previous ones.

Let us now define

$$!_{\circ}\underline{A} = (A, (\mathsf{comp}(R_A))^* \cdot R_A),$$

i.e. $!\!\!\circ$ is an impoverished, sequential, version of ! where a new "thread" of R_A can be started only when the previous one is completed. Obviously, $R_{!\!\circ\underline{A}} \subseteq R_{!\underline{A}}$.

An important case of $!\!\circ\underline{A}$, which we use in the following, is when $\underline{A}$ is well-opened, i.e. each play in R_A can contain only one occurrence of an initial move, namely, the first move of the play (all games interpreting ICA types are of that kind). Then $!\!\circ\underline{A}$ contains plays which might have many occurrences of initial moves, but only one occurrence of an initial question can be open (pending) at any time. Similarly, $\bigotimes_{1\leq i\leq n} !\!\circ\underline{A}$ contains plays with at most n pending questions; we shall write $\underline{A}^n$ for it. We use this construction to specify restricted function spaces: instead of $\underline{A} \Rightarrow \underline{B} = !\underline{A} \multimap \underline{B}$ we consider $\underline{A}^n \multimap \underline{B}$. These restrictions turn out to give the correct interpretation of the bounds inferred by the type system for SCC.

Regardless of whether we deal with standard ICA type or typing judgements (annotated with bounds or not) $[\![\cdots]\!]$ stands for the usual interpretation in $\mathcal{G}_{\mathrm{sat}}$ (i.e. the information about bounds is completely ignored by $[\![\cdots]\!]$). We introduce the notation $[\![\cdots]\!]_\eta$ for bound-sensitive semantic interpretation.

Let $\Gamma \vdash_r M : \theta$, where $\Gamma = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. In $\mathcal{G}_{\mathrm{sat}}$ it is standardly interpreted by a strategy for the game $[\![\Gamma \vdash \theta]\!] = [\![\theta_1]\!] \times \ldots \times [\![\theta_k]\!] \Rightarrow [\![\theta]\!]$ or, equivalently, $![\![\theta_1]\!] \otimes \ldots \otimes ![\![\theta_k]\!] \multimap [\![\theta]\!]$. Suppose η represents a vector of resource bounds consistent with $\Gamma \vdash_r M : \theta$. It is not necessary that η includes all the bounds used in the resource-sensitive type judgement. Then the corresponding bounded game, denoted by $[\![\Gamma \vdash \theta]\!]_\eta$, is defined inductively in the same way as $[\![\Gamma \vdash \theta]\!]$ except that whenever a bound n is specified by η (for an occurrence of $\rightarrow$ or θ_i), we use $A^n \multimap B$ and A^n instead of respectively $A \Rightarrow B = !A \multimap B$ and $!A$.

Example 13 *Suppose we have* $x_1 : (\mathbf{com}^9 \rightarrow \mathbf{sem})^5, x_2 : (\mathbf{exp}^3 \rightarrow \mathbf{com})^7 \vdash M : \mathbf{exp}^7 \rightarrow \mathbf{var}$. *The complete vector of resource bounds is* $(9, 5, 3, 7, 7)$. *Let* η *stand for the distinguished bounds* $(-, 5, 3, -, 7)$. *Then*

$$[\![\mathbf{com} \rightarrow \mathbf{sem}, \mathbf{exp} \rightarrow \mathbf{com} \vdash \mathbf{exp} \rightarrow \mathbf{var}]\!]_\eta$$
$$= (![\![\mathbf{com}]\!] \multimap [\![\mathbf{sem}]\!])^5 \otimes !([\![\mathbf{exp}]\!]^3 \multimap [\![\mathbf{com}]\!]) \multimap ([\![\mathbf{exp}]\!]^7 \multimap [\![\mathbf{var}]\!])$$

This notation is flexible enough to handle assumes, guarantees or combined assume-guarantee resource bounds in a uniform way.

Now we are ready to interpret the bounds given by the type system using the game model. We denote the interpretation by $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$. It is simply $[\![\Gamma \vdash M : \theta]\!]$ in which O-moves are restricted to those allowed by the $A^n \multimap B$ games consistent with the bounds in η^{a}:

$$\boxed{[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}} = [\![\Gamma \vdash M : \theta]\!] \cap R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathrm{a}}}}.}$$

More precisely, for each occurrence m of an initial move from such B Opponent will not be allowed to play an initial move from A justified by m if the current position already contains n pending questions justified by m. The guaranteed bounds given by SCC are then sound in that they are correct approximations of the shape of positions explored by P when O behaves according to η^{a}, i.e. the positions are not only in $R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathrm{a}}}}$ but also in $R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$, where by $\eta\eta'$ we mean the two combined constraint vectors.

Theorem 14 $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}} \subseteq R_{[\![\Gamma \vdash \theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$.

PROOF. The theorem can be proved by induction on the derivation of $\Gamma \vdash_r M : \theta$. As before, application is, technically, the most difficult. In all other cases it is easy to see that the definition of $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$ is compositional: $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$ can be defined directly by induction on the structure of $\vdash_r$ derivations. Consequently, a simple appeal to the induction hypothesis does the job.

For application, let $\eta_1^{\mathrm{a}}, \eta_2^{\mathrm{a}}, \eta_3^{\mathrm{a}}$ represent the assumed bounds of the respective three judgements:

$$\frac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n\Delta \vdash_r MN : \theta'}.$$

Let us make the following definitions:

$$\begin{array}{ll} \sigma_1 = [\![\Gamma \vdash M : \theta \to \theta']\!]_{\eta_1^{\mathrm{a}}} & \sigma_1' = [\![\Gamma \vdash M : \theta \to \theta']\!]_{\eta_3^{\mathrm{a}} \restriction \Gamma, \theta'} \\ \sigma_2 = [\![\Delta \vdash N : \theta]\!]_{\eta_2^{\mathrm{a}}} & \sigma_2' = [\![\Delta \vdash N : \theta]\!]_{\eta_3^{\mathrm{a}} \restriction \Delta}, \end{array}$$

where we write $\eta \restriction \Theta$ for η restricted to a list of types Θ. The only difference between σ_i and σ_i' $(i = 1, 2)$ is that in σ_i' there are no bounds on O-moves in θ. Otherwise, they are subject to the same restrictions, because $\eta_1^{\mathrm{a}} \restriction \Gamma, \theta' = \eta_3^{\mathrm{a}} \restriction \Gamma, \theta'$ and $\eta_2^{\mathrm{a}} \restriction \Delta = \eta_3^{\mathrm{a}} \restriction \Delta$.

Consider $[\![\Gamma, n\Delta \vdash MN : \theta']\!]_{\eta_3^{\mathrm{a}}}$, which is defined by interactions of σ_1' with σ_2'. Note that up to the first move in θ , σ_1' behaves in the same way as σ_1. Then, as σ_1' and σ_2' interact, the induction hypotheses imply that the guarantees provided by each of the strategies match the assumed bounds of the other $(\eta_1^{\mathrm{g}} \restriction \theta = \eta_2^{\mathrm{a}} \restriction \theta$, $\eta_2^{\mathrm{g}} \restriction \theta = \eta_1^{\mathrm{a}} \restriction \theta)$. Thus, the interaction of σ_1' and σ_2' is actually constrained to positions of σ_1 and σ_2. Consequently, $[\![\Gamma, n\Delta \vdash MN : \theta']\!]_{\eta_3^{\mathrm{a}}}$ can also be defined compositionally by interactions of σ_1 and σ_2 and the rest easily follow. $\Box$

The sets of complete plays induced by the restricted denotations $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^{\mathrm{a}}})$ turn out to provide a fully abstract model of $\sqsubseteq\!\!\!\approx_r$.

Lemma 15 *Suppose* $\Gamma \vdash_r M_1, M_2 : \theta$ *and let* η^{a} *be the final assignment of assumed bounds. Then* $\mathsf{comp}(\llbracket \Gamma \vdash_r M_1 : \theta \rrbracket_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}(\llbracket \Gamma \vdash_r M_2 : \theta \rrbracket_{\eta^{\mathrm{a}}})$ *implies* $\Gamma \vdash M_1 \sqsubseteq\!\!\!\approx_r M_2 : \theta$.

PROOF. Suppose $\vdash_r C[M_i] : \mathbf{com}$ (i=1,2) and $C[M_1] \Downarrow$. Then, by the soundness of $\mathcal{G}_{\mathrm{sat}}$[16], $\mathsf{comp}(\llbracket C[M_1] \rrbracket) \neq \emptyset$. As noted in the proof of Theorem 14, $\llbracket C[M_1] \rrbracket$ can be defined inductively through $\llbracket \Gamma \vdash M_1 \rrbracket_{\eta^{\mathrm{a}}}$ so, because $\mathsf{comp}(\llbracket \Gamma \vdash_r M_1 \rrbracket_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}(\llbracket \Gamma \vdash_r M_2 \rrbracket_{\eta^{\mathrm{a}}})$, we also have $\mathsf{comp}(\llbracket C[M_2] \rrbracket) \neq \emptyset$. Thus, again by the adequacy of $\mathcal{G}_{\mathrm{sat}}$, $C[M_2] \Downarrow$ and indeed $M_1 \sqsubseteq\!\!\!\approx_r M_2$. $\square$

To prove the converse we need to strengthen the definability result from [34] to ensure that terms corresponding to positions are also typable. This means that we cannot simply regard justification pointers as indicating parallel threads of computation and have to sequentialize threads where possible. The details of the adaptation are presented in Appendix A. The example below illustrates this new definability algorithm.

Example 16 *Let us consider a position in the game for* $\mathbf{com}^2 \to \mathbf{com}$*:*

$$run \cdot run_1 \cdot run_1 \cdot ok_1 \cdot run_1 \cdot ok_1 \cdot run_1 \cdot ok_1 \cdot ok_1 \cdot ok.$$

The algorithm from [16,34] would return

$$\lambda x.\mathbf{newvar}\ x_0, x_3, x_5, x_7, x_8 := 0\ \mathbf{in}\ x_0 := 1; M;\ WAIT_9,$$

where $M \equiv (P_1 \,||\, P_2 \,||\, P_4 \,||\, P_6)$ *and*

$$\begin{array}{ll} P_1 \equiv WAIT_1; x; x_3 := 1 & P_2 \equiv WAIT_2; x; x_5 := 1, \\ P_4 \equiv WAIT_4; x; x_8 := 1, & P_6 \equiv WAIT_6; x; x_7 := 1, \end{array}$$

but the term does not have the required type $\mathbf{com}^2 \to \mathbf{com}$*. The refined version produces* $M \equiv (P_1; P_4) \,||\, (P_2; P_6)$ *instead. The term* $WAIT_i$ *tests whether all variables* x_j *with indices less than* i *are set to* 1 *and diverges if they are not.*

Consequently, the following properties can be proved as in [16].

Lemma 17 *Suppose* θ *is a type with constraints* η *and* $s \in R_{\llbracket \theta \rrbracket_\eta}$*. Then there exists a term* $\vdash_r M : \theta$ *such that* $\llbracket M \rrbracket$ *is the least saturated strategy containing* s*.*

Lemma 18 *Suppose* $\Gamma \vdash_r M_1, M_2 : \theta$ *and let* η^{a} *be the final assignment of assumed bounds.* $\Gamma \vdash M_1 \sqsubseteq\!\!\!\approx_r M_2 : \theta$ *implies* $\mathsf{comp}(\llbracket \Gamma \vdash_r M_1 : \theta \rrbracket_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}(\llbracket \Gamma \vdash_r$

$M_2 : \theta\rrbracket_{\eta^{\mathrm{a}}}$).

Lemmas 15 and 18 imply full abstraction.

Theorem 19 *Suppose* $\Gamma \vdash_r M_1, M_2 : \theta$ *and let* η^{a} *be the final assignment of assumed bounds.*

- $\Gamma \vdash M_1 \sqsubseteq_{\approx r} M_2 : \theta \iff \mathsf{comp}(\llbracket \Gamma \vdash_r M_1 : \theta\rrbracket_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}(\llbracket \Gamma \vdash_r M_2 : \theta\rrbracket_{\eta^{\mathrm{a}}})$.
- $\Gamma \vdash M_1 \cong_r M_2 : \theta \iff \mathsf{comp}(\llbracket \Gamma \vdash_r M_1 : \theta\rrbracket_{\eta^{\mathrm{a}}}) = \mathsf{comp}(\llbracket \Gamma \vdash_r M_2 : \theta\rrbracket_{\eta^{\mathrm{a}}})$.

6 Regular Representation

In this section we show sets of complete plays $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta\rrbracket_{\eta^{\mathrm{a}}})$ can be represented faithfully as regular languages and compared by checking language equivalence. The main difficulty to be addressed is the need to represent pointers.

For any bounded game θ, we represent the positions of $R_{\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ using the alphabet $\mathcal{A}(\theta)$ defined as follows:

$$\begin{aligned} \mathcal{A}(\beta) &= M_{\llbracket\beta\rrbracket}, \\ \mathcal{A}(\gamma \to \theta) &= \mathcal{A}(\gamma) + \mathcal{A}(\theta), \\ \mathcal{A}(\theta^n) &= \{\, m^i \mid m \in \mathcal{A}(\theta),\ 1 \le i \le n \,\}. \end{aligned}$$

Thus, elements of $\mathcal{A}(\theta)$ can be seen as moves of $\llbracket\theta\rrbracket$ decorated with a vector $\vec{i} = (i_1, \ldots, i_k)$ of labels produced by the last clause. The letters $m^{\vec{i}}$ will be used to encode occurrences of m in positions from $R_{\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ subject to two invariants.

- If a question q has several open occurrences then each of them will be represented by a different vector.
- Suppose an occurrence of a question q be represented by $q^{\vec{i}}$. If an occurrence of another question m is justified by the above occurrence of q, then m is represented as $m^{j\vec{i}}$ for some $j \in \mathbb{N}$.

We explain below how each position from the game under question will be represented so that the invariants are satisfied and only letters from $\mathcal{A}(\theta)$ are used. Note that the initial moves of θ occur without labels in $\mathcal{A}(\theta)$. They will also be represented as such in positions (this never leads to ambiguities since positions have unique initial moves). Given a representation of s a representation of sm is calculated as follows.

- If m is an answer to an occurrence of q in s represented by $q^{\vec{i}}$ then m is represented by $m^{\vec{i}}$.

- If m is a question justified by an occurrence of q in s represented by $q^{\vec{i}}$, then there exists a sub-game $G^n_m \multimap G_q$ of $[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}$ such that q, m are initial moves of respectively G_q, G_m. Since sm is a position of $[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}$ there can be at most $n-1$ open questions in s that are justified by the same occurrence of q and, hence, represented by $q^{j\vec{i}}$. Thus one of the labels from $\{\,1,\ldots,n\,\}$, say k, has not been used. Then we represent m as $m^{k\vec{i}}$ (any such k will do).

Note that, thanks to the labels, justification pointers can be uniquely reconstructed from the representation, so it is faithful. However, it is not unique because of the arbitrary choice of k. We will say that a representation is *canonical* if k is always chosen to be the least k available. The notion of canonicity is crucial to comparing representations of positions as they will provide the link between language equivalence and program equivalence.

Given a set S of strings over $\mathcal{A}(\theta)$ representing a set of plays (e.g. a strategy) on $R_{[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ we write $\mathsf{can}(S)$ for the canonization of that representation.

Lemma 20 *If S is regular so is $\mathsf{can}(S)$.*

PROOF. Given an automaton accepting S one construct one for $\mathsf{can}(S)$. The number of open questions in any position of $R_{[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ is uniformly bounded. Hence, with the help of finite memory we can keep track of all labels of open questions during the runtime of the automaton and relabel the accepted letters as required in a canonical representation. Since only finite store is needed, all this can be done by a finite automaton, so $\mathsf{can}(S)$ is also regular. The formal construction proceeds by annotating the states of the original automaton with all possible configurations of the finite memory. A possible version is shown below.

Let $\langle\, Q, z_0, \delta, F \,\rangle$ be the automaton accepting S. Then we define $\langle\, Q', z'_0, \delta', F' \,\rangle$ as follows. Let $\mathcal{M} = \{\, (m, \vec{i}, \vec{j}) \mid m^{\vec{i}}, m^{\vec{j}} \in \mathcal{A}(\theta),\ m \text{ is a question} \,\}$ and $Q' = Q \times \wp_B(\mathcal{M})$ where $\wp_B(\mathcal{M})$ stands for the set of subsets of $\mathcal{M}$ of size at most B and B is the uniform bound on the number of open questions in $R_{[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$. Let $z'_0 = (z_0, \emptyset)$ and $F' = F \times \{\, \emptyset \,\}$.

- If $\delta(z, q) = z'$ then define $\delta'((z, \emptyset), q) = (z', \{(q, (), ())\})$
- If $\delta(z, a^{\vec{i}}) = z'$ then for all $q, \vec{h}, X \in \wp_B(\mathcal{M})$ such that $(q, \vec{i}, \vec{h}) \in X$ and $q \vdash a$ include $(z', X \setminus \{\, (q, \vec{i}, \vec{h}) \,\})$ in $\delta'((z, X), a^{\vec{h}})$.
- If $\delta(z, q^{j\vec{i}}) = z'$ then for all $q_1, \vec{h}, X \in \wp_B(\mathcal{M})$ such that $(q_1, \vec{i}, \vec{h}) \in X$ and $q_1 \vdash q$ include (z', X') in $\delta'((z, X), q^{k\vec{h}})$ provided $\{\,1,\ldots,k-1\,\} \subseteq U$, $k \notin U$, where $U = \{\, u \mid \exists_{\vec{g}, q_2}((q_2, \vec{g}, u\vec{h}) \in X \text{ and } q_1 \vdash q_2) \,\}$, $X' = X \cup \{\, (q, j\vec{i}, k\vec{h}) \,\}$ and $|X'| \leq B$. $\Box$

Theorem 21 *The canonical representation of $\mathsf{comp}([\![\Gamma \vdash_r M : \theta]\!]_{\eta^{\mathrm{a}}})$, denoted*

simply by $[\![\Gamma \vdash_r M : \theta]\!]$ *below, is a regular language over*

$$\mathcal{A} = \mathcal{A}(\theta_1^{n_1}) + \ldots + \mathcal{A}(\theta_k^{n_k}) + \mathcal{A}(\theta).$$

PROOF. Many of the definitions for the imperative part of the language have the same flavour as those for Idealized Algol [9]. Sometimes the operation on regular languages will have to be followed by an explicit conversion to canonical form.

We define the $(\!|\Gamma \vdash M|\!)$ notations by the following decompositions:

$$\begin{aligned}
[\![\Gamma \vdash_r M : \mathbf{com}]\!] &= run \cdot (\!|\Gamma \vdash_r M|\!) \cdot ok \\
[\![\Gamma \vdash_r M : \mathbf{exp}]\!] &= \sum_{i=0}^{\mathrm{MAX}} q \cdot (\!|\Gamma \vdash_r M|\!)_i \cdot i \\
[\![\Gamma \vdash_r M : \mathbf{var}]\!] &= \sum_{i=0}^{\mathrm{MAX}} write(i) \cdot (\!|\Gamma \vdash_r M|\!)_i^w \cdot ok + \sum_{i=0}^{\mathrm{MAX}} read \cdot (\!|\Gamma \vdash_r M|\!)_i^r \cdot i \\
[\![\Gamma \vdash_r M : \mathbf{sem}]\!] &= grab \cdot (\!|\Gamma \vdash_r M|\!)^g \cdot ok + release \cdot (\!|\Gamma \vdash_r M|\!)^r \cdot ok
\end{aligned}$$

It is convenient to define $[\![\Gamma \vdash_r M]\!]$ via $(\!|\Gamma \vdash_r M|\!)$:

$$\begin{aligned}
(\!|\Gamma \vdash_r M; N|\!) &= (\!|\Gamma \vdash_r M|\!) \cdot (\!|\Gamma \vdash_r N|\!) \\
(\!|\Gamma \vdash_r \mathbf{if}\ M\ \mathbf{then}\ N_1\ \mathbf{else}\ N_2|\!) &= (\!|\Gamma \vdash_r M|\!)_0 \cdot (\!|\Gamma \vdash_r N_2|\!) + (\sum_{i=1}^{\mathrm{MAX}} (\!|\Gamma \vdash_r M|\!)_i) \cdot (\!|\Gamma \vdash_r N_1|\!) \\
(\!|\Gamma \vdash_r \mathbf{while}\ M\ \mathbf{do}\ N|\!) &= ((\sum_{i=1}^{\mathrm{MAX}} (\!|\Gamma \vdash_r M|\!)_i) \cdot (\!|N|\!))^* \cdot (\!|\Gamma \vdash_r M|\!)_0 \\
(\!|\Gamma \vdash_r !M|\!)_i &= (\!|\Gamma \vdash_r M|\!)_i^r \\
(\!|\Gamma \vdash_r M := N|\!) &= \sum_{i=0}^{\mathrm{MAX}} ((\!|\Gamma \vdash_r N|\!)_i \cdot (\!|\Gamma \vdash_r M|\!)_i^w) \\
(\!|\Gamma \vdash_r \mathbf{grab}(M)|\!) &= (\!|\Gamma \vdash_r M|\!)^g \\
(\!|\Gamma \vdash_r \mathbf{release}(M)|\!) &= (\!|\Gamma \vdash_r M|\!)^r
\end{aligned}$$

The above cases do not require explicit canonization. Neither does that of λ-abstraction which is interpreted using the appropriate associativity isomorphism of the disjoint sum.

For semaphore or variable binding it suffices to consider the histories in which the moves occur completely sequentially (in a canonical representation they are labelled with 1) [16]. We define

$$\mathsf{cell}_m = (read^1 \cdot m^1)^* \cdot (\sum_{i=0}^{\mathrm{MAX}} (write(i)^1 \cdot ok^1 \cdot (read^1 \cdot i^1)^*))^*$$

and

$$[\![\Gamma \vdash_r \mathbf{newvar}\ x := m\ \mathbf{in}\ M : \beta]\!] = ([\![\Gamma, x : \mathbf{var}^n \vdash M]\!] \cap \overline{\mathsf{cell}_m}) \setminus \mathcal{A}(\mathbf{var}^n),$$

where $\overline{E} = E \amalg (\mathcal{A}(\Gamma) + \mathcal{A}(\beta))^*$ and $L \setminus A$ is obtained by erasing the symbols from A in strings from L. Similarly, let us define $G = \mathit{grab}^1 \cdot \mathit{ok}^1$ and $R = \mathit{release}^1 \cdot \mathit{ok}^1$. Then

$$\begin{aligned}
&[\![\Gamma \vdash_r \mathbf{newsem}\ x := 0\ \mathbf{in}\ M : \beta]\!] = \\
&\qquad ([\![\Gamma, x : \mathbf{sem}^m \vdash M]\!] \cap \overline{(G \cdot R)^* \cdot (G + \epsilon)}) \setminus \mathcal{A}(\mathbf{sem}^m) \\
&[\![\Gamma \vdash_r \mathbf{newsem}\ x := 1\ \mathbf{in}\ M : \beta]\!] = \\
&\qquad ([\![\Gamma, x : \mathbf{sem}^m \vdash_r M]\!] \cap \overline{(R \cdot G)^* \cdot (R + \epsilon)}) \setminus \mathcal{A}(\mathbf{sem}^m).
\end{aligned}$$

We can take $(\!|\Gamma, \Delta \vdash_r M \,||\, N|\!)$ to be $(\!|\Gamma \vdash_r M|\!) \amalg (\!|\Delta \vdash_r N|\!)$, which preserves canonicity.

Contraction is defined through renaming of labels associated with y. The labels $1, \ldots, n$ are replaced with $m+1, \ldots, m+n$. This induces a homomorphism on the language so the result is still regular but needs canonization.

We write id_θ for $[\![x : \theta \vdash_r x : \theta]\!]$. $\mathrm{id}_{\mathbf{com}}$ is defined by $\{\, \mathit{run} \cdot \mathit{run}^1 \cdot \mathit{ok}^1 \cdot \mathit{ok} \,\}$. For other base types the definition is analogous [9]. We extend it to function types $\theta^n \to \theta'$ as follows. Let $\mathrm{id}_{\theta'} = \sum_{q,a}(q \cdot q^1 \cdot \mathrm{id}^{q,a}_{\theta'} \cdot a^1 \cdot a)$. Then

$$\mathrm{id}_{\theta^n \to \theta'} = \mathsf{can}(\sum_{q,a}(q \cdot q^1 \cdot (\amalg_{i=1}^n (\mathrm{id}^{i1}_\theta)^*\ \amalg\ \mathrm{id}^{q,a}_{\theta'}) \cdot a^1 \cdot a)),$$

where id^{j1}_θ is id_θ in which each move $m^{\vec{i}}$ is replaced with $m^{\vec{i}j1}$ if it comes from the right copy of θ and with $m^{\vec{i}j}$ if it comes from the left one.

For application it is crucial that canonical representations interact as the interaction has to be represented in the same way both by the function and by the argument. Let $\Delta = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. For $i = 1, \ldots, n$ let $\tilde{N}_i$ be the same as $[\![\Delta \vdash_r N : \theta]\!]$ except that the moves from the θ-component are additionally decorated with the label i while the original labels of moves from θ_j $(1 \le j \le k)$ (i.e. $1, \ldots, n_j$) are replaced respectively with $(i-1)n_j + 1, \ldots, in_j$. Clearly, these operations preserve regularity. Then we can define $[\![\Gamma, \Delta \vdash MN : \theta']\!]$ to be $\mathsf{can}((\tilde{M} \cap \tilde{N}) \setminus \mathcal{A}(\theta^n))$ where

$$\begin{aligned}
\tilde{M} &= [\![\Gamma \vdash M : \theta^n \to \theta']\!] \amalg \mathcal{A}(\theta_1^{n \cdot n_1})^* \amalg \cdots \amalg \mathcal{A}(\theta_k^{n \cdot n_k})^* \\
\tilde{N} &= \mathcal{A}(\Gamma)^* \amalg \mathsf{can}(\amalg_{i=1}^n (\tilde{N}_i)^*) \amalg \mathcal{A}(\theta')^*.
\end{aligned}$$

Finally, no changes are needed to interpret subsumption. $\square$

Theorem 22 *$\sqsubseteq_{\sim r}$ and $\cong_r$ are decidable.*

7 Further Work

We have already stated that we plan to study the syntactic properties of the system separately. The previous section establishes that there is a finite-state representation of terms of SCC, and that it can be used, in principle, for model checking using a method similar to [10]. Lemma 10 and the various examples we give suggest that the restrictions imposed by the tighter typing discipline are not onerous. However, to claim a fully automated verification (and certification) procedure the issue of automated type inference must be investigated. Finally, only by incorporating these theoretical results in a model-checking tool can we evaluate the practicality of the method.

References

[1] S. Abramsky, R. Jagadeesan, P. Malacaria, Full abstraction for PCF, Information and Computation 163 (2000) 409–470.

[2] J. M. E. Hyland, C.-H. L. Ong, On Full Abstraction for PCF: I. Models, observables and the full abstraction problem, II. Dialogue games and innocent strategies, III. A fully abstract and universal game model, Information and Computation 163(2) (2000) 285–408.

[3] S. Abramsky, G. McCusker, Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions, in: P. W. O'Hearn, R. D. Tennent (Eds.), Algol-like languages, Birkhaüser, 1997, pp. 297–329.

[4] S. Abramsky, G. McCusker, Full abstraction for Idealized Algol with passive expressions, Theoretical Computer Science 227 (1999) 3–42.

[5] J. Laird, Full abstraction for functional languages with control, in: Proceedings of 12th IEEE Symposium on Logic in Computer Science, 1997, pp. 58–67.

[6] S. Abramsky, K. Honda, G. McCusker, Fully abstract game semantics for general references, in: Proceedings of IEEE Symposium on Logic in Computer Science, Computer Society Press, 1998, pp. 334–344.

[7] P. Malacaria, C. Hankin, Generalized flowcharts and games (extended abstract), in: Proceedings of ICALP 1998, Vol. 1443 of Lecture Notes in Computer Science, Springer-Verlag, 1998, pp. 363–374.

[8] P. Malacaria, C. Hankin, Non-deterministic games and program analysis: an application to security, in: Proceedings of 14th IEEE Symposium on Logic in Computer Science, IEEE, 1999, pp. 443–452.

[9] D. R. Ghica, G. McCusker, Reasoning about Idealized Algol using regular expressions, in: Proceedings of ICALP, Vol. 1853 of Lecture Notes in Computer Science, Springer-Verlag, 2000, pp. 103–115.

[10] S. Abramsky, D. R. Ghica, A. S. Murawski, C.-H. L. Ong, Applying game semantics to compositional software modelling and verification, in: Proceedings of TACAS, Vol. 2988 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 421–435.

[11] C.-H. L. Ong, Observational equivalence of 3rd-order Idealized Algol is decidable., in: Proceedings of IEEE Symposium on Logic in Computer Science, Computer Society Press, 2002, pp. 245–256.

[12] D. R. Ghica, Regular-language semantics for a call-by-value programming language, in: Proceedings of MFPS, Vol. 45 of Electronic Notes in Computer Science, Elsevier, 2001.

[13] A. S. Murawski, Functions with local state: regularity and undecidability, submitted for publication, 41pp. (2004).

[14] D. R. Ghica, A regular-language model for Hoare-style correctness statements, in: Proceedings of the Verification and Computational Logic Workshop, Florence, Italy, 2001.

[15] D. R. Ghica, A games-based foundation for compositional software model checking, Ph.D. thesis, Queen's University School of Computing, Kingston, Ontario, Canada, also available as Oxford University Computing Laboratory Research Report RR-02-13 (November 2002).

[16] D. R. Ghica, A. S. Murawski, Angelic semantics of fine-grained concurrency, in: Proceedings of FOSSACS, Vol. 2987 of Lecture Notes in Computer Science, Springer-Verlag, 2004, pp. 211–225. An extended version is available as [34]

[17] S. Abramsky, Beyond full abstraction: model-checking for Algol-like languages, 2001, series of lectures given at the Marktoberdorf Summer School.

[18] R. Alur, T. A. Henzinger, O. Kupferman, Alternating-time temporal logic, Journal of the ACM 49 (5) (2002) 672–713.

[19] M. Hofmann, Linear types and non-size-increasing polynomial time computation, in: Proceedings of the 14th Symposium on Logic in Computer Science, IEEE, 1999, pp. 464–473.

[20] M. Hofmann, A type system for bounded space and functional in-place update, Nordic Journal of Computing 7 (4) (2002) 258–289.

[21] J. Hughes, L. Pareto, Recursion and dynamic data-structures in bounded space: Towards embedded ML programming, ACM SIGPLAN Notices 34 (9) (1999) 70–81, proceedings of the ICFP.

[22] M. Tofte, Region inference for higher-order functional languages, in: Proceedings of SAS, Vol. 983 of Lecture Notes in Computer Science, 1995, pp. 19–20.

[23] A. Mycroft, R. Sharp, A statically allocated parallel functional language, in: Proceedings of ICALP, Vol. 1853 of Lecture Notes in Computer Science, 2000, pp. 37–48.

[24] G. C. Necula, Proof-carrying code, in: Proceedings of the 24th ACM Symposium on Principles of Programming Languages, 1997, pp. 106–119.

[25] K. Wansbrough, S. L. P. Jones, Once upon a polymorphic type, in: Proceedings of the 26th ACM Symposium on Principles of Programming Languages, 1999, pp. 15–28.

[26] J. C. Reynolds, Syntactic control of interference, in: Proceedings of POPL, 1978, pp. 39–46.

[27] J. T. Udding, A formal model for defining and classifying delay-insensitive circuits and systems, Distributed Computing 1(4) (1986), 197–204.

[28] H. Jifeng, M. B. Josephs, and C. A. R. Hoare, A theory of synchrony and asynchrony. In Programming Concepts and Methods. Elsevier, 1990, pp. 459–473.

[29] J. Laird, A games semantics for idealized CSP. In Proceedings of the 17th Annual Conference on Mathematical Foundations of Programming Semantics, (Aarhus, Denmark, May 2001), Electronic notes in Theoretical Computer Science, Elsevier, pp. 157–176.

[30] M. L. Minsky, Computation: Finite and Infinite Machines, Prentice-Hall, 1967.

[31] H. P. Barendregt, Lambda calculi with types, in: S. Abramsky, D. M. Gabbay, T. S. E. Maibaum (Eds.), Background: Computational Structures, Vol. 2 of Handbook of Logic in Computer Science, Oxford University Press, Oxford, England, 1992, pp. 117–309.

[32] S. Brookes, The essence of Parallel Algol, in: Proceedings of the 11th Symposium on Logic in Computer Science, 1996, pp. 164–173, also published as Chapter 21 of [35].

[33] G. McCusker, Games for recursive types, BCS Distinguished Dissertation, Cambridge University Press, 1998.

[34] D. R. Ghica, A. S. Murawski, Angelic semantics of fine-grained concurrency, Tech. Rep. PRG-RR-03-20, Oxford University Computing Laboratory, available from `http://users.ox.ac.uk/~coml0074/papers/asfgc.pdf` (2003).

[35] P. W. O'Hearn, R. D. Tennent (Eds.), ALGOL-like Languages, Progress in Theoretical Computer Science, Birkhäuser, Boston, 1997, two volumes.

A Resource-Sensitive Definability

We define a recursive algorithm, called $PROC^+$, which takes a position s in $R_{[\![\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ and returns a term $\vdash_r P_s : \theta$ such that $[\![P_s]\!]$ is the least saturated

strategy containing s. $PROC^+$ relies on a recursive procedure $PROC$ which takes the original position as the initial argument. In the recursive invocations of $PROC$, the argument is a subsequence of the form $s \restriction m$, where $t \restriction m$ is the subsequence of t consisting of m and all moves hereditarily justified by m, always an O-question. Note that consequently a move in t is answered in t iff it is answered in s.

Throughout the execution of $PROC$ it is convenient to use indices relative to the original s; we write s_i for the ith move of s, assuming s_0 initial. In order to generate the desired position we need to control the way in which both P and O move. We control P-moves using guards that wait for special side-effects (time-stamps) caused by O-moves. The effects take place only if a correct O-move is played and we make sure that they occur only once by using a fresh semaphore for each O-move. This allows us to enforce arbitrary synchronization policies, restricting the order of moves present in the original sequence up to the reorderings dictated by the saturation conditions. To that effect, a *global* variable x_j, i.e. a variable which is bound by **new** at the top level and initialized to 0, is associated which each index of an O move in s. The time-stamp consists of assigning 1 to the variable, $x_j := 1$.

For $1 \leq j \leq |s| - 1$, let us define:

$$O_j = \{\, i \in \mathbb{N} \mid 0 \leq i < j,\ s_i \text{ is an O-move} \,\}.$$

We define $WAIT_j$ as the term which checks for time-stamps originating from all the O-moves with indices smaller than j:

$$WAIT_j \equiv [\bigwedge_{g \in O_j} (!x_g = 1)].$$

$PROC(t : \theta)$ where $\theta = \theta_1^{n_1} \to \ldots \to \theta_h^{n_h} \to \beta$ is defined as follows in two stages which manage O-questions and P-answers, and respectively P-questions and O-answers.

If t is empty, $\lambda p_1 \cdots p_h.\Omega_{\theta_0}$ is returned. Otherwise, let $o = s_i$ be the initial move of t (which is always an O-question).

(1) For $a = 1, \ldots, h$ let $i_1^a < \cdots < i_{m_a}^a$ be the s-indices of all occurrences of questions from θ_a explicitly justified by s_i. We define a function $\phi : \{\, i_1^a, \ldots, i_{m_a}^a \,\} \to \{\, 1, \ldots, n_a \,\}$ (which will assign moves to threads) inductively using the ordering $i_1^a < \cdots < i_{m_a}^a$. Consider i_c^a. Then at most $n_a - 1$ out of $s_{i_1^a}, \ldots, s_{i_{c-1}^a}$ are open in $s_{\leq i_c^a}$. Let Q_c^{op} be the set containing them (i.e. $|Q_c^{\mathrm{op}}| < n_a$). Define $\phi(i_c^a)$ to be the least number from $\{\, 1, \ldots, n_a \,\}$ different from each $\phi(m)$ ($m \in Q_c^{\mathrm{op}}$). Assume we have terms $P_{i_d^a} : \mathbf{com}$ ($d = 1, \ldots, m_a$) to be defined later. For $k = 1, \ldots, n_a$ let R_a^k be the *sequential* composition of all $P_{i_d^a}$ such that $\phi(i_d^a) = k$ (ordered in the same

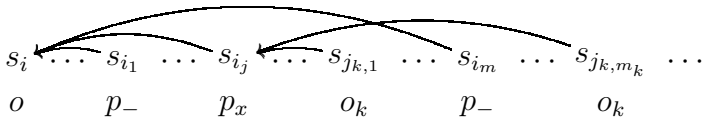

Fig. A.1. Questions and justification pointers

way as i_d^a's). Then let $R_a = R_a^1 \,||\, \cdots \,||\, R_a^{n_a}$. *PROC* returns the following results, depending on β.

- $\beta = \mathbf{com}$:

$$\lambda p_1 \cdots p_h.(x_i := 1); (R_1 \,||\, \cdots \,||\, R_h); PANS_i^{\mathbf{com}}$$

where

$$PANS_i^{\mathbf{com}} \equiv \begin{cases} \Omega_{\mathbf{com}} & s_i \text{ is unanswered in } t \\ WAIT_{i'}\ s_{i'} & s_{i'} \text{ answers } s_i \text{ in } t \end{cases}$$

By convention, $(R_1 \,||\, \cdots \,||\, R_h)$ degenerates to **skip** for $m = 0$.

- $\beta = \mathbf{exp}$: Same as for **com** except that $PANS_i^{\mathbf{com}}$ is replaced with $PANS_i^{\mathbf{exp}}$ defined below.

$$PANS_i^{\mathbf{exp}} \equiv \begin{cases} \Omega_{\mathbf{exp}} & s_i \text{ is unanswered in } t \\ WAIT_{i'}; s_{i'} & s_{i'} \text{ answers } s_i \text{ in } t \end{cases}$$

- $\beta = \mathbf{var}$:
  - If $s_i = \mathit{read}$:

$$\mathbf{mkvar}(\,\lambda x.\Omega_{\mathbf{com}}, (x_i := 1); (R_1 \,||\, \cdots \,||\, R_h); PANS_i^{\mathbf{exp}}).$$

  - If $s_i = \mathit{write}(v)$:

$$\mathbf{mkvar}(\lambda x.\mathbf{if}\,(x = v)\,\mathbf{then}\,x_i := 1; (R_1 \,||\, \cdots \,||\, R_h); PANS_i^{\mathbf{com}}, \Omega_{\mathbf{exp}})\,.$$

    The presence of the $x = v$ test serves to ensure that the only acceptable move by O is only that which writes v, and no other value.

- $\beta = \mathbf{sem}$ is analogous to **var**:
  - If $s_i = \mathit{grab}$:

$$\mathbf{mksem}((x_i := 1); (R_1 \,||\, \cdots \,||\, R_h); PANS_i^{\mathbf{com}}, \Omega_{\mathbf{com}}).$$

  - If $s_i = \mathit{release}$:

$$\mathbf{mksem}(\Omega_{\mathbf{com}}, (x_i := 1); (R_1 \,||\, \cdots \,||\, R_h); PANS_i^{\mathbf{com}}).$$

(2) Let $i_1 < \cdots < i_m$ be the s-indices of all occurrences of questions justified by s_i. Here we show how to define the terms P_{i_j} for $1 \le j \le m$. Let us fix j and suppose that $s_{i_j} = p_x$ $(1 \le x \le h)$ and $\theta_x = {\theta'_1}^{m_1} \to \ldots \to {\theta'_n}^{m_n} \to$

β'. Let $o_1, \ldots, o_n$ be all the O-questions enabled by p_x (corresponding to $\theta'_1, \ldots, \theta'_n$ respectively).

For each k $(1 \leq k \leq n)$ let $j_{k,1} < \cdots < j_{k,m_k}$ be the s-indices of all occurrences of o_k in t which are explicitly justified by s_{i_j} (see Figure A.1).

If $m_k = 0$, then $P^k_j \equiv \Omega_{\theta'_k}$. Otherwise, for all $l = 1, \ldots, m_k$ we make the following definitions: $P^{k,l}_j \equiv PROC(t \restriction s_{j_{k,l}} : \theta'_k)$ and

$$P^k_j \equiv ONCE_{w_{j_{k,1}}}[P^{k,1}_j] \textbf{ or } \cdots \textbf{ or } ONCE_{w_{j_{k,m_k}}}[P^{k,m_k}_j],$$

where $w_{j_{k,1}}, \ldots, w_{j_{k,m_k}}$ are fresh semaphore names and $ONCE_w(M) = \textbf{grab}(w); M$. Finally, we define the terms P_{i_j}, depending on β'. The fresh variables z_c are used to "store" O-answers for future tests.

First, it is useful to define the following macros:

$$OANS^{\textbf{com}}_c \equiv \begin{cases} \textbf{skip} & s_c \text{ is unanswered in } t \\ x_{c'} := 1 & s_{c'} \text{ answers } s_c \text{ in } t \end{cases}$$

$$OANS^{\textbf{exp}}_c \equiv \begin{cases} \textbf{skip} & s_c \text{ is unanswered in } t \\ \textbf{if } (!z_c = s_{c'}) \textbf{ then } x_{c'} := 1 \textbf{ else skip} & s_{c'} \text{ answers } s_c \text{ in } t \end{cases}$$

- For $\beta' = \textbf{com}$, $P_{i_j} \equiv WAIT_{i_j}; (p_x P^1_j \cdots P^n_j); OANS^{\textbf{com}}_{i_j}$
- For $\beta' = \textbf{exp}$, $P_{i_j} \equiv WAIT_{i_j}; z_{i_j} := (p_x P^1_j \cdots P^n_j); OANS^{\textbf{exp}}_{i_j}$.
- For $\beta' = \textbf{var}$ there are two sub-cases:
  - If $s_{i_j} = read$, $P_{i_j} \equiv WAIT_{i_j}; z_{i_j} := !(p_x P^1_j \cdots P^n_j); OANS^{\textbf{exp}}_{i_j}$.
  - If $s_{i_j} = write(v)$, $P_{i_j} \equiv WAIT_{i_j}; (p_x P^1_j \cdots P^n_j) := v; OANS^{\textbf{com}}_{i_j}$.
- For $\beta' = \textbf{sem}$, P_{i_j} there are two sub-cases:
  - If s_{i_j} is $grab$, $P_{i_j} \equiv WAIT_{i_j}; \textbf{grab}(p_x P^1_j \cdots P^n_j); OANS^{\textbf{com}}_{i_j}$
  - If s_{i_j} is $release$, $P_{i_j} \equiv WAIT_{i_j}; \textbf{release}(p_x P^1_j \cdots P^n_j); OANS^{\textbf{com}}_{i_j}$

After $PROC(s : \theta)$ returns $\lambda p_1 \cdots p_k.M$, all variables and semaphores used in the construction of M (i.e. x_-, z_-, w_-) must be bound at the topmost level (the variables x_- must be initialized to 0, the semaphores w_- to 0, the initial values of z_- are irrelevant). For $\beta = \textbf{com}, \textbf{exp}$ this is done by taking

$$\lambda p_1 \cdots p_k.\textbf{newvar}\, \vec{x}, \vec{z} := \vec{0}\, \textbf{in}\, (\textbf{newsem}\, \vec{w} := \vec{0}\, \textbf{in}\, M).$$

For $\beta = \textbf{var}, \textbf{com}$ the binders have to be pushed inside **mkvar** or **mksem**. We denote the final term by $PROC^+(s : \theta)$.