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
{drg,andrzej,lo}@comlab.ox.ac.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, admits 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 [1–5]. Ghica and McCusker [6] 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 [7]. Ghica has subsequently extended the approach to a call-by-value language with arrays [8], Hoare-style assertions [9] and specification [10].

In this paper we propose a game-based framework for compositional model checking of concurrent programs. We have developed a fully-abstract game model for ICA, a concurrent language based on Idealized Algol extended with parallel command composition ($C \,||\, C$) and binary semaphores (**sem**) manipulated by blocking primitives **grab** and **release** [11]. However, the model seems unsuitable 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

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

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* (SRIA), a type system introduced by Abramsky to identify higher-order terms of a sequential language denotable by "pointer-free" finitary game models [12].

The bounds imposed on the number of pending questions by SCC can be seen as a kind of *assume-guarantee* reasoning (see e.g. [13]): 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 [14] or space [15]; applications of such systems are as diverse as execution in embedded systems [16], memory management [17], compilation to hardware [18] or proof-carrying code [19]. Type systems have also been used to control more abstract resources, such as variable usage for improved compilation [20] or interference effects for specification and verification [21].

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 [11]. The main thrust of the paper is thus semantic; we plan to investigate the type-theoretic issues of SCC separately.

2 SCC: A Resource-Bounding Type System

Theorem 1. *May-equivalence of second-order ICA terms is undecidable.*

Proof. Using the game semantic model of [11] we can show that observational equivalence of terms in the second-order fragment of ICA can be reduced to the halting problem for Minsky machines, which is known to be undecidable [22]. □

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 thus for the recursion-free fragment with **while**-loops. Divergence, $\Omega_{\mathbf{com}}$, can then be defined to be $\mathbf{while}\,1\,\mathbf{do}\,\mathbf{skip}$. 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* (resp. *guarantee*) if it occurs in the left-hand scope of an even (resp. odd) number of $\to$'s in θ. Formally, m is an assume (a guarantee) in θ iff $\theta = \mathcal{A}[m]$ $(\theta = \mathcal{G}[m])$:

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

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

Assumes concern the behaviour of the program context and guarantees 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:

$$\beta \leq \beta \qquad \frac{m \leq n}{\theta^n \leq \theta^m} \qquad \frac{\gamma_2 \leq \gamma_1 \quad \theta_1 \leq \theta_2}{\gamma_1 \to \theta_1 \leq \gamma_2 \to \theta_2}.$$

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 Fig. 1. 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 "linear" rules, in which the context are required to be disjoint, as opposed to the rules for sequential operators ($\Box$ can stand for $;$, $:=$, etc.) including branching and iteration. The contraction rule has been modified so that the assumed bounds on the contracted variable are accumulated into the new variable.

SCC enjoys the standard syntactic properties of a typed lambda calculus (basis, generation, subterm, substitution and subject reduction lemmas) [23].

Remark 1. The rule for application is also "linear," requiring disjoint environments for the function and the application. The reason is that call-by-name application is a straitjacketed 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 function, any

$$x : \theta^1 \vdash_r x : \theta \qquad \frac{\Gamma \vdash_r M : \theta \qquad \theta \leq \theta'}{\Gamma \vdash_r M : \theta'}$$

$$\frac{\Gamma \vdash_r M : \theta}{\Gamma, x : \gamma \vdash_r M : \theta} \qquad \frac{\Gamma, x : \theta^m, y : \theta^n \vdash_r M : \theta'}{\Gamma, x : \theta^{m+n} \vdash_r M[x/y] : \theta'}$$

$$\frac{\{\Gamma \vdash_r M_1 : \theta_1\} \qquad \Gamma \vdash_r M_2 : \theta_2}{\Gamma \vdash_r \{M_1\} \,\square\, M_2 : \theta_3} \qquad \frac{\Gamma \vdash_r C_1 : \mathbf{com} \qquad \Delta \vdash_r C_2 : \mathbf{com}}{\Gamma, \Delta \vdash_r C_1 \,||\, C_2 : \mathbf{com}}$$

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

$$\frac{\Gamma \vdash_r M : \mathbf{exp} \qquad \Gamma \vdash_r M_1, M_2 : \beta}{\Gamma \vdash_r \mathbf{if}\, M\, \mathbf{then}\, M_1\, \mathbf{else}\, M_2 : \beta} \qquad \frac{\Gamma \vdash_r B : \mathbf{exp} \qquad \Gamma \vdash_r C : \mathbf{com}}{\Gamma \vdash_r \mathbf{while}\, B\, \mathbf{do}\, C : \mathbf{com}}$$

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

Fig. 1. SCC typing rules

computation arising in an application $F(M)$ also arises in the parallel composition $\cdots F(\cdots)\cdots \,||\, \cdots M \cdots$, where the ellipses stand for code manipulating semaphores so that the right interleaving of effects is enforced [11]. The restriction of application to disjoint environments is also used in SRIA [12].

Example 1. For any $n \in \mathbb{N}$ we have

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

Not all ICA terms are typable in SCC. For example, the application of the term 5 to term 4 above is untypable, even using subsumption. However, as seen in Lemma 1 below, there is a wide class of typable terms.

Given the bounds for environment, SCC can be used to certify bounds for the program. We say that a term $\Gamma \vdash M : \theta$ is r-typable if for any assignment of assumes to Γ and θ 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. Therefore our approach will not apply to all higher-order terms. The Lemma below, which is by no means exhaustive, shows that there is a rich supply of terms which are r-typable.

Lemma 1. *Any term with β-redexes of order at most 1 (i.e. the argument is of base type or a first-order function) is r-typable. In particular, β-normal terms of all types are r-typable.*

Using SCC we can define a new *observational approximation* relation $\sqsubseteq_r$ 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 using (possibly several copies of) the given type derivation of $\Gamma \vdash_r M_i$, up to appropriate renaming of variables. We define $\Gamma \vdash_r M_1 \sqsubseteq_r M_2$ 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 $\Gamma \vdash_r M_1 \sqsubseteq_r M_2$ and $\Gamma \vdash_r M_2 \sqsubseteq_r M_1$. 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 $\sqsubseteq_r$ we require $\Gamma \vdash_r M_i : \theta$ to have the same annotations. But 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 2 ([24]). 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 Sec. 4.

3 The Game Model

In [11] we have given a game model which is fully abstract for $\sqsubseteq$ and $\cong$ (the standard notions of observational approximation and equivalence in ICA). We use the model to interpret the annotations from the type system and to show how the model can be used to reason about $\sqsubseteq_r, \cong_r$.

We give a sketch of the model. 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 $\lambda_A^{OP}(m) = P$.

An arena is called *flat* if its questions are all initial (consequently the P-moves can only be answers). In arenas used to interpret base types all questions are initial and P-moves answer them as detailed in the table below.

Arena	O-question	P-answers	Arena	O-question	P-answers
$[\![\mathbf{com}]\!]$	*run*	*ok*	$[\![\mathbf{exp}]\!]$	q	$0, \ldots, N$
$[\![\mathbf{var}]\!]$	*read*	n	$[\![\mathbf{sem}]\!]$	*grab*	*ok*
	$write(n)$	*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 and m_A a move from M_A. Not all justified sequences are considered "valid" plays:

Definition 1. *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 $s' = \cdots\ q \cdots\ m$ of s, the question q must be pending before m is played.*

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

For two shuffled sequences s_1 and s_2, $s_1 \amalg s_2$ denote 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 $\bigcup_{i \in \mathbb{N}} X^i$.

Definition 2. *A* strategy *σ on A (written $\sigma : A$) is a prefix-closed subset of P_A, which is* O-complete *(i.e. if $s \in \sigma$ and $so \in P_A$, where o is an (occurrence of an) O-move, then $so \in \sigma$).*

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 and 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 lhs of $\preceq$ have the same justifier as on the rhs.

Definition 3. *A strategy σ is* saturated *iff $s \in \sigma$ and $s' \preceq s$ imply $s' \in \sigma$.*

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

;	$q_1\, run\, ok\, q_0\, a_0\, a_1$	$\vert\vert$	$run_2\, run_0\, run_1\, ok_0\, ok_1\, ok_2$
:=	$run_2\, q_1\, n_1\, write(n)_0\, ok_0\, ok_2$	!	$q\, read\, n\, n$
grab	$run_1\, grab_0\, ok_0\, ok_1$	**release**	$run_1\, release_0\, ok_0\, ok_1$

newvar $X := n$	$q\, q\, (read\, n)^*\, (\sum_{i=0}^{N}(write(i)\ ok\ (read\ i)^*))^*\, a\, a$
newsem $S := 0$	$q\, q\, (grab\ ok\ release\ ok)^*\, (grab\ ok + \epsilon)\, a\, a$
newsem $S := 1$	$q\, q\, (release\ ok\ grab\ ok)^*\, (release\ ok + \epsilon)\, a\, a.$

As shown in [11], $\mathcal{G}_{\mathrm{sat}}$ is fully abstract for $\cong$ in the sense mentioned below. Let $\mathsf{comp}(\sigma)$ be the set of non-empty complete plays of a strategy σ.

Theorem 2. $\Gamma \vdash M_1 \lesssim M_2 \iff \mathsf{comp}([\![\Gamma \vdash M_1]\!]) \subseteq \mathsf{comp}([\![\Gamma \vdash M_2]\!])$.

4 The Game Model Revisited

In order to analyze the positions induced by terms in more detail we shall define a more restricted 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. [25]).

Definition 4. *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 $R_A = P_A$. Bounded games can be combined using a number of constructions.

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

and $\underline{A} \Rightarrow \underline{B} = !\underline{A} \multimap \underline{B}$. 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 $!_s\underline{A} = (A, (\mathsf{comp}(R_A))^* \cdot R_A)$, i.e. $R_{!_s\underline{A}} \subseteq R_{!\underline{A}}$ and $!_s$ is an impoverished, sequential, version of ! where a new "thread" of R_A can be started only when the previous one is completed. An important case of $!_s\underline{A}$ 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 $!_s\underline{A}$ 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, $\bigotimes_{1\leq i\leq n} !_s\underline{A}$ 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 $A \Rightarrow B = {!A} \multimap B$ we consider $A^n \multimap B$. These restrictions turn out to give the correct interpretation of the bounds inferred by the type system given before.

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 3. 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

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

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. Let us define the restriction $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$ of the semantics according to the assumed bounds to be $[\![\Gamma \vdash M : \theta]\!] \cap R_{[\![\Gamma\vdash\theta]\!]_{\eta^{\mathrm{a}}}}$, i.e. $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$ is $[\![\Gamma \vdash M : \theta]\!]$ in which O-moves are restricted to those allowed by the $A^n \multimap B$ games consistent with the bounds in η^{a}. 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 constrain vectors.

Theorem 3. $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}} \in R_{[\![\Gamma\vdash\theta]\!]_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$

The theorem can be proved by induction on the derivation of $\Gamma \vdash_r M : \theta$ in conjunction with the Lemma below, which validates the induction (note that the original definition of $[\![\Gamma \vdash M : \theta]\!]_{\eta^{\mathrm{a}}}$ relies on the full interpretation $[\![\Gamma \vdash M : \theta]\!]$).

Lemma 2. *The definition of* $[\![\Gamma \vdash_r M : \theta]\!]_{\eta^{\mathrm{a}}}$ *is compositional:* $[\![\Gamma \vdash_r M : \theta]\!]_{\eta^{\mathrm{a}}}$ *can be defined directly by induction on the structure of* $\vdash_r$ *derivations. Moreover, the inductive clauses are identical to those for* $[\![\cdots]\!]$.

Crucially, given $\dfrac{\Gamma \vdash_r M : \theta^n \to \theta' \qquad \Delta \vdash_r N : \theta}{\Gamma, n\Delta \vdash_r MN : \theta'}$ where $\eta_1^{\mathrm{a}}, \eta_2^{\mathrm{a}}, \eta_3^{\mathrm{a}}$ represent the assumed bounds of the respective three judgements, in order to calculate $[\![\Gamma, n\Delta \vdash_r MN : \theta']\!]_{\eta_3^{\mathrm{a}}}$ one only needs positions from $[\![\Gamma \vdash_r M : \theta^n \to \theta']\!]_{\eta_1^{\mathrm{a}}}$ and $[\![\Delta \vdash_r N : \theta]\!]_{\eta_2^{\mathrm{a}}}$. The above is an important step in our method, because it allows the inductive definition of restricted denotations: full denotations are much more complicated that the resource-restricted ones. The sets of complete plays induced by the restricted denotations $\mathsf{comp}([\![\Gamma \vdash_r M : \theta]\!]_{\eta^{\mathrm{a}}})$ provide a fully abstract model of $\sqsubseteq_r$.

Lemma 3. *Suppose* $\Gamma \vdash_r M_1, M_2 : \theta$ *and let* η^{a} *be the final assignment of assumed bounds. Then* $\mathsf{comp}([\![\Gamma \vdash_r M_1 : \theta]\!]_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2 : \theta]\!]_{\eta^{\mathrm{a}}})$ *implies* $\Gamma \vdash M_1 \sqsubseteq_r M_2 : \theta$.

Proof. Suppose $\vdash_r C[M_i] : \mathbf{com}$ (i=1,2) and $C[M_1] \Downarrow$. Then by the soundness of $\mathcal{G}_{\mathrm{sat}}$ [11] $\mathsf{comp}([\![C[M_1]]\!]) \neq \emptyset$. By Lemma 2 $[\![C[M_1]]\!]$ can be defined inductively through $[\![\Gamma \vdash M_1]\!]_{\eta^{\mathrm{a}}}$, so because $\mathsf{comp}([\![\Gamma \vdash_r M_1]\!]_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2]\!]_{\eta^{\mathrm{a}}})$ we also have $\mathsf{comp}([\![C[M_2]]\!]) \neq \emptyset$. Thus again, by the adequacy of $\mathcal{G}_{\mathrm{sat}}$ $C[M_2] \Downarrow$, so indeed $M_1 \sqsubseteq_r M_2$. $\square$

To prove the converse we need to strengthen the definability result from [11] 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. Below we illustrate the difference between the two definability algorithms.

Example 4. 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 [11] would return $\lambda x.\mathbf{newvar}\ x_0, x_3, x_5, x_7, x_8 := 0\ \mathbf{in}\ x_0 := 1; M; \mathit{WAIT}_9$, where $M \equiv (P_1 \,||\, P_2 \,||\, P_4 \,||\, P_6)$, $P_1 \equiv \mathit{WAIT}_1; x; x_3 := 1$, $P_2 \equiv \mathit{WAIT}_2; x; x_5 := 1$, $P_4 \equiv \mathit{WAIT}_4; x; x_8 := 1$, $P_6 \equiv \mathit{WAIT}_6; x; x_7 := 1$, 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.

Lemma 4. *Suppose* θ *is a type with constraints* η *and* $s \in R_{[\![\theta]\!]_\eta}$. *Then there exists a term* $\vdash_r M : \theta$ *such that* $[\![M]\!]$ *is the least saturated strategy containing* s.

Theorem 4. *Using the same assumes as above.* $\Gamma \vdash M_1 \sqsubseteq_r M_2 : \theta$ *implies* $\mathsf{comp}([\![\Gamma \vdash_r M_1 : \theta]\!]_{\eta^{\mathrm{a}}}) \subseteq \mathsf{comp}([\![\Gamma \vdash_r M_2 : \theta]\!]_{\eta^{\mathrm{a}}})$.

5 Regular Representation

In this section we show how 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 an alphabet $\mathcal{A}(\theta)$ defined as follows: $\mathcal{A}(\beta) = M_{\llbracket\beta\rrbracket}$, $\mathcal{A}(\gamma \to \theta) = \mathcal{A}(\gamma) + \mathcal{A}(\theta)$ and $\mathcal{A}(\theta^n) = \{\, m^i \mid m \in \mathcal{A}(\theta),\ 1 \le i \le n \,\}$. Thus, elements of $\mathcal{A}(\theta)$ can be seen as moves of $\llbracket\theta\rrbracket$ decorated with a vector $\vec{i} = (i_1, \dots, i_k)$ 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. Let 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 represented by $q^{\vec{i}}$ then m is represented as $m^{\vec{i}}$.
- If m is a question justified by an occurrence of q represented as $q^{\vec{i}}$, then there exists a sub-game $G_m^n \multimap G_q$ of $\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}$ such that q, m are initial moves of respectively G_q, G_m. Since sm is a position of $\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}$ there can be at most $n-1$ open questions in s that are justified by the same occurrence of q and, hence, represented as $q^{j\vec{i}}$ in s. Thus one of the labels from $\{1, \dots, n\}$, 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 $R_{\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ we write $\mathsf{can}(S)$ for the canonization of that representation.

Lemma 5. *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 $R_{\llbracket\theta\rrbracket_{\eta^{\mathrm{a}}\eta^{\mathrm{g}}}}$ 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. $\square$

Let $\Gamma = \theta_1^{n_1}, \ldots, \theta_k^{n_k}$. We will show that the canonical representation of $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^a})$, which we denote simply by $\llbracket \Gamma \vdash_r M : \theta \rrbracket$, is a regular language over $\mathcal{A} = \mathcal{A}(\theta_1^{n_1}) + \ldots + \mathcal{A}(\theta_k^{n_k}) + \mathcal{A}(\theta)$. Many of the definitions, especially for the imperative part of the language have the same flavour as those for Idealized Algol [6] so we focus on the more difficult cases below. Sometimes the operation on regular languages will have to be followed by an explicit conversion to canonical form.

Let $(\!|\Gamma, \Delta \vdash_r C : \mathbf{com}|\!)$ be defined by $\llbracket \Gamma, \Delta \vdash_r C \rrbracket = run \cdot (\!|\Gamma, \Delta \vdash_r C|\!) \cdot ok$. Then we take $(\!|\Gamma, \Delta \vdash_r C_1 \,||\, C_2|\!)$ to be $(\!|\Gamma \vdash_r C_1|\!) \amalg (\!|\Delta \vdash_r C_2|\!)$ (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 define $\mathrm{id}_{\mathsf{com}}$ by $\{ run \cdot run^1 \cdot ok^1 \cdot ok \}$. For other base types the definition is analogous. We extend it to function types $\theta^n \to \theta'$ as follows. Let $\mathrm{id}_{\theta'} = \sum_{q,a}(q \cdot q^1 \cdot \mathrm{id}_{\theta'}^{q,a} \cdot a^1 \cdot a)$. Then $\mathrm{id}_{\theta^n \to \theta'} = \mathsf{can}(\sum_{q,a}(q \cdot q^1 \cdot (\amalg_{i=1}^n (\mathrm{id}_\theta^{i1})^* \amalg \mathrm{id}_{\theta'}^{q,a}) \cdot a^1 \cdot a))$, where id_θ^{j1} is id_θ in which each move $m^{\vec{i}}$ is replaced with $m^{\vec{i}j1}$.

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 $\llbracket \Delta \vdash_r N : \theta \rrbracket$ except that the moves from the θ-component are additionally decorated with the label i while the original labels of moves from θ_j $(1 \leq j \leq k)$ (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 $\llbracket \Gamma, \Delta \vdash MN : \theta' \rrbracket$ to be $\mathsf{can}((\tilde{M} \cap \tilde{N}) \setminus \mathcal{A}(\theta^n))$ where $\tilde{N} = \mathcal{A}(\Gamma)^* \amalg \mathsf{can}(\amalg_{i=1}^n (\tilde{N}_i)^*) \amalg \mathcal{A}(\theta')^*$ and $\tilde{M} = \llbracket \Gamma \vdash M : \theta^n \to \theta' \rrbracket \amalg \mathcal{A}(\theta_1^{n \cdot n_1})^* \amalg \ldots \amalg \mathcal{A}(\theta_k^{n \cdot n_k})^*$. Finally, no changes are needed to interpret subsumption.

Theorem 5. *$\llbracket \Gamma \vdash_r M \rrbracket$ is a canonical representation of $\mathsf{comp}(\llbracket \Gamma \vdash_r M : \theta \rrbracket_{\eta^a})$.*

Theorem 6. *$\precsim_r$ and $\cong_r$ are decidable.*

6 Further Work

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 [7]. Lemma 1 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 (FDR seems a good candidate [26]) can we evaluate the practicality of the method.

References

1. Abramsky, S., Jagadeesan, R., Malacaria, P.: Full abstraction for PCF. Information and Computation **163** (2000)
2. Hyland, J.M.E., Ong, C.-H.L.: On full abstraction for PCF: I, II and III. Information and Computation **163** (2000)
3. Abramsky, S., McCusker, G.: Linearity, sharing and state. In: Proceedings of 1996 Workshop on Linear Logic. ENTCS **3**., Elsevier (1996)
4. Laird, J.: Full abstraction for functional languages with control. In: LICS 12. (1997)
5. Abramsky, S., Honda, K., McCusker, G.: A fully abstract game semantics for general references. In: LICS 13. (1998)
6. Ghica, D.R., McCusker, G.: Reasoning about Idealized ALGOL using regular languages. In: ICALP 27. LNCS **1853**. (2000)
7. Abramsky, S., Ghica, D.R., Murawski, A.S., Ong, C.-H.L.: Applying game semantics to compositional software modeling and verification. In: TACAS 10. LNCS **2988** (2004)
8. Ghica, D.R.: Regular language semantics for a call-by-value programming language. In: Proceedings of MFPS 17. ENTCS **45** (2001)
9. Ghica, D.R.: A regular-language model for Hoare-style correctness statements. In: Proceedings of the Verification and Computational Logic (2001)
10. Ghica, D.R.: A Games-based Foundation for Compositional Software Model Checking. PhD thesis, Queen's University School of Computing, Canada (2002)
11. Ghica, D.R., Murawski, A.S.: Angelic semantics of fine-grained concurrency. In: FOSSACS 7. LNCS **2987** (2004)
12. Abramsky, S.: Beyond Full Abstraction: model-checking for Algol-like languages Marktoberdorf International Summer School 2001. (lecture slides)
13. Alur, R., Henzinger, T.A., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49** (2002) 672–713
14. Hofmann, M.: Linear types and non-size-increasing polynomial time computation. In: LICS 14 (1999)
15. Hofmann, M.: A type system for bounded space and functional in-place update. Nordic Journal of Computing **7** (2000)
16. Hughes, J., Pareto, L.: Recursion and dynamic data-structures in bounded space: Towards embedded ML programming. ICFP 4. ACM SIGPLAN Notices **34** (1999)
17. Tofte, M.: Region inference for higher-order functional languages. LNCS **983** (1995)
18. Mycroft, A., Sharp, R.: A statically allocated parallel functional language. In: ICALP 27. LNCS **1853** (2000)
19. Necula, G.C.: Proof-carrying code. In: POPL 24 (1997)
20. Wansbrough, K., Jones, S.L.P.: Once upon a polymorphic type. In: POPL 26 (1999)
21. Reynolds, J.C.: Syntactic control of interference. In: POPL 5 (1978)
22. Minsky, M.: Computation: Finite and Infinite Machines. Prentice Hall (1967)
23. Barendregt, H.P.: Lambda calculi with types. In Abramsky, S., Gabbay, D.M., Maibaum, T.S.E., eds.: Background: Computational Structures. Vol. 2 of Handbook of Logic in Computer Science. Oxford University Press (1992)
24. Brookes, S.: The essence of Parallel Algol. In: LICS 11 (1996)
25. McCusker, G.: Games and Full Abstraction for a Functional Metalanguage with Recursive Types. Distinguished Dissertations. Springer-Verlag Limited (1998)
26. Roscoe, W.A.: Theory and Practice of Concurrency. Prentice-Hall (1998)