CSP Representation of Game Semantics for Second-order Idealized Algol*

Aleksandar Dimovski and Ranko Lazić

Department of Computer Science
University of Warwick
Coventry CV4 7AL, UK
{aleks,lazic}@dcs.warwick.ac.uk

Abstract. We show how game semantics of an interesting fragment of Idealised Algol can be represented compositionally by CSP processes. This enables observational equivalence and a range of properties of terms-in-context (i.e. open program fragments) to be checked using the FDR tool. We have built a prototype compiler which implements the representation, and initial experimental results are positive.

1 Introduction

Context. One of the main breakthroughs in theoretical computer science in the past decade has been the development of game semantics (e.g. [10, 1]). Types are modelled by games between Player (i.e. term) and Opponent (i.e. context or environment), and terms are modelled by strategies. This has produced the first accurate (i.e. fully abstract and fully complete) models for a variety of programming languages and logical systems.

It has recently been shown that, for several interesting programming language fragments, game semantics yields algorithms for software model checking. The focus has been on Idealised Algol (IA) [13] with active expressions. IA is similar to Core ML. It is a compact programming language which combines the fundamental features of imperative languages with a full higher-order procedure mechanism. For example, simple forms of classes and objects may be encoded in IA.

For second-order recursion-free IA with iteration and finite data types, [7] shows that game semantics can be represented by regular expressions, so that observational equivalence between any two terms can be decided by equality of regular languages. For third order and without iteration, it was established in [12] that game semantics can be represented by deterministic pushdown automata, which makes observational equivalence decidable by equality of deterministic context-free languages. Classes of properties other than observational equivalence can also be checked algorithmically, such as language containment or Hoare triples (e.g. [2, 8]).

* We acknowledge support by the EPSRC (GR/S52759/01). The first author was also supported by the Intel Corporation.

In recent years, software model checking has become an active research area, and powerful tools have been built (e.g. [4]). Compared with other approaches to software model checking, the approach based on game semantics has a number of advantages [3]:

- there is a model for any term-in-context, which enables verification of program fragments which contain free variable and procedure names;
- game semantics is compositional, which facilitates verifying a term to be broken down into verifying its subterms;
- terms are modelled by how they interact with their environments, and details of their internal state during computations are not recorded, which results in small models.

Our contribution. In this paper, we show how game semantics of second-order recursion-free IA with iteration and finite data types can be represented in the CSP process algebra. For any term-in-context, we compositionally define a CSP process whose terminated traces are exactly all the complete plays of the strategy for the term. Observational equivalence between two terms can then be decided by checking two traces refinements between CSP processes.

Compared with the representation by regular expressions (or automata) [7], the CSP representation brings several benefits:

- CSP operators preserve traces refinement (e.g. [16]), which means that a CSP process representing a term can be optimised and abstracted compositionally at the syntactic level (e.g. using process algebraic laws), and its set of terminated traces will be preserved or enlarged;
- the ProBE and FDR tools [6] can be used to explore CSP processes visually, to check traces refinements automatically, and to debug interactively when traces refinements do not hold;
- compositional state-space reduction algorithms in FDR [15] enable smaller models to be generated before or during refinement checking;
- composition of strategies, which is used in game semantics to obtain the strategy for a term from strategies for its subterms, is represented in CSP by renaming, parallel composition and hiding operators, and FDR is highly optimised for verification of such networks of processes;
- parameterised terms (as a simple example, a program which reverses an array of values of an arbitrary data type α) can be interpreted by single parameterised processes, which can then be verified e.g. using techniques from the infinite-state model checking literature.

We have implemented a prototype compiler which, given any IA term-in-context, outputs a CSP process representing its game semantics. We report some initial experimental results, which show that for model generation, FDR outperforms the tool based on the representation by regular expressions [9].

Organisation In the next section, we present the fragment of IA we are addressing. Section 3 contains brief introductions to game semantics, CSP and

FDR. In section 4, we define the CSP representation of game semantics for the IA fragment. Correctness of the CSP model, and decidability of observational equivalence by traces refinement, are shown in section 5. We present the experimental results in section 6. Finally, in section 7, we conclude and discuss future work.

2 The programming language

Idealized Algol [13] is a functional-imperative language with usual imperative features as iteration, branching, assignment, sequential composition, combined with a function mechanism based on a typed call-by-name lambda calculus. We consider only the recursion-free second-order fragment of this language. We will only work with finite data sets.

The language has basic data types τ, which are a finite subset of the integers and the booleans. The phrase types of the language are expressions, commands and variables, plus first-order function types.

$$\begin{aligned} \tau &::= \mathit{int} \mid \mathit{bool} \\ \sigma &::= \mathit{exp}[\tau] \mid \mathit{comm} \mid \mathit{var}[\tau] \\ \theta &::= \sigma \mid \sigma \times \sigma \times \cdots \sigma \to \sigma \end{aligned}$$

Terms are introduced using type judgements of the form:

$$\Gamma \vdash M : \theta, \text{ where } \Gamma = \{\iota_1 : \theta_1, \cdots, \iota_k : \theta_k\}$$

For the sake of simplicity, we assume that terms are β-normal, so there is no λ abstractions, and also function application is restricted to free identifiers. The terms of the language and their typing rules are given in Table 1.

For type $\mathit{exp}[\mathit{int}]$, the finitary fragment contains constants n belonging to a finite subset of the set of integers, and for type $\mathit{exp}[\mathit{bool}]$ there are constants *true* and *false*. For type *comm*, there are basic commands *skip*, to do nothing, and *diverge* which causes a program to enter an unresponsive state similar to that caused by an infinite loop. The other commands are assignment to variables, $V := E$, conditional operation, $\mathit{if}\ B\ \mathit{then}\ C\ \mathit{else}\ C'$, and while loop, $\mathit{while}\ B\ \mathit{do}\ C$. Also, we have sequential composition of commands $C \mathbin{;} C'$ as well as sequential composition of a command with an expression or a variable. There are also term formers for dereferencing variables, !V, application of first-order free identifiers to arguments $\iota M_1 \cdots M_k$, and local variable declaration $\mathit{new}[\tau]\ \iota\ \mathit{in}\ C$. Finally, we have a function definition *let* constructor.

3 Background

3.1 Game semantics

We give an informal overview of game semantics and we illustrate it with some examples. A more complete introduction can be found in [2].

Table 1. Terms and typing rules

$$\frac{}{\Gamma \vdash \mathit{true} : \mathit{exp}[\mathit{bool}]} \qquad \frac{}{\Gamma \vdash n : \mathit{exp}[\mathit{int}]}$$

$$\frac{}{\Gamma \vdash \mathit{skip} : \mathit{comm}} \qquad \frac{}{\Gamma \vdash \mathit{diverge} : \mathit{comm}}$$

$$\frac{}{\Gamma, \iota : \theta \vdash \iota : \theta} \qquad \frac{\Gamma \vdash V : \mathit{var}[\tau]}{\Gamma \vdash !V : \mathit{exp}[\tau]}$$

$$\frac{\Gamma \vdash E_1 : \mathit{exp}[\mathit{int}] \quad \Gamma \vdash E_2 : \mathit{exp}[\mathit{int}]}{\Gamma \vdash E_1 + E_2 : \mathit{exp}[\mathit{int}]} \qquad \frac{\Gamma \vdash E_1 : \mathit{exp}[\mathit{int}] \quad \Gamma \vdash E_2 : \mathit{exp}[\mathit{int}]}{\Gamma \vdash E_1 = E_2 : \mathit{exp}[\mathit{bool}]}$$

$$\frac{\Gamma \vdash B_1 : \mathit{exp}[\mathit{bool}] \quad \Gamma \vdash B_2 : \mathit{exp}[\mathit{bool}]}{\Gamma \vdash B_1 \ \mathit{and} \ B_2 : \mathit{exp}[\mathit{bool}]} \qquad \frac{\Gamma \vdash B : \mathit{exp}[\mathit{bool}]}{\Gamma \vdash \mathit{not} \ B : \mathit{exp}[\mathit{bool}]}$$

$$\frac{\Gamma \vdash V : \mathit{var}[\tau] \quad \Gamma \vdash E \ : \ \mathit{exp}[\tau]}{\Gamma \vdash V := E : \mathit{comm}} \qquad \frac{\Gamma \vdash C : \mathit{comm} \quad \Gamma \vdash M : \sigma}{\Gamma \vdash C \mathbin{⨟} M : \sigma}$$

$$\frac{\Gamma \vdash B : \mathit{exp}[\mathit{bool}] \quad \Gamma \vdash M_1 : \sigma \quad \Gamma \vdash M_2 : \sigma}{\Gamma \vdash \mathit{if} \ B \ \mathit{then} \ M_1 \ \mathit{else} \ M_2 : \sigma} \qquad \frac{\Gamma \vdash B : \mathit{exp}[\mathit{bool}] \quad \Gamma \vdash C : \mathit{comm}}{\Gamma \vdash \mathit{while} \ B \ \mathit{do} \ C : \mathit{comm}}$$

$$\frac{\Gamma \vdash \iota : \sigma_1 \times \cdots \times \sigma_k \to \sigma \quad \Gamma \vdash M_i : \sigma_i}{\Gamma \vdash \iota(M_1, M_2, \ldots, M_k) \ : \ \sigma} \qquad \frac{\Gamma, \iota : \mathit{var}[\tau] \vdash C : \mathit{comm}}{\Gamma \vdash \mathit{new}[\tau] \ \iota \ \mathit{in} \ C : \mathit{comm}}$$

$$\frac{\Gamma, \iota : \sigma_1 \times \cdots \times \sigma_k \to \sigma' \vdash M : \theta \qquad \Gamma, \iota_1 : \sigma_1, \ldots, \iota_k : \sigma_k \vdash N : \sigma'}{\Gamma \vdash \mathit{let} \ \iota \ (\iota_1 : \sigma_1, \ldots, \iota_k : \sigma_k) \ = \ N \ \mathit{in} \ M : \theta}$$

As the name suggests, game semantics models computation as a certain kind of game, with two participants, called Player(P) and Opponent(O). P represents the term (program), while O represents the environment, i.e. the context in which the term is used. A play between O and P consists of a sequence of moves, governed by rules. For example, O and P need to take turn and every move needs to be justified by a preceding move. The moves are of two kinds, questions and answers.

To every type in the language corresponds a game — the set of all possible plays (sequences of moves). A term is represented as a set of all complete plays in the appropriate game, more precisely as a strategy for that game — a predetermined way for P to respond to O's moves.

For example, in the game for the type $\mathit{exp}[\tau]$, there is an initial move q and corresponding to it a single response to return its value. So a complete play for a constant $\vdash \ c \ : \ \mathit{exp}[\tau]$ is:

O: q (opponent asks for value)
P: c (player answers to the question)

Consider a more complex example $\iota : \mathit{exp}[\mathit{int}] \to \mathrm{exp}[\mathit{int}] \ \vdash \ \iota(2) : \mathit{exp}[\mathit{int}]$, where the identifier ι is some non-locally defined function.
A play for this term begins with O asking for the value of the result expression by playing the question move q, and P replies asking for the returned value of the non-local function ι, move q^{ι}. In this situation, the function ι may need to evaluate its argument, represented by O's move $q^{\iota 1}$ — what is the value of the

first argument to ι. P will respond with answer $2^{\iota 1}$. Here, O could repeat the question $q^{\iota 1}$ to represent the function which evaluates its argument more than once. In the end, when O plays the move n^{ι} — the value returned from ι, P will copy this value and answer to the first question with n.
A sample complete play for this term, when the function ι evaluates only once its argument, is:

O: q (asks for result value)
P: q^{ι} (P asks for value returned from function ι)
O: $q^{\iota 1}$ (O questions what is the first argument to ι)
P: $2^{\iota 1}$ (P answers: 2)
O: n^{ι} (O supplies the value returned from ι)
P: n (P gives the answer to the first question)

In the game for commands, there is an initial move *run* to initiate a command, and a single response *done* to signal termination of the command. Thus the only complete play for the command $\vdash$ *skip* : *comm* is:

O: *run* (start executing)
P: *done* (terminate command)

Variables are represented as objects with 2 methods: the "read method" for dereferencing, represented by an initial move *read*, to which a response could be any element of τ, and the "write method", for assignment, represented by an initial move $write(x)$ for any element x of τ, to which there is only one possible response *ok*. For example, a complete play for the command $v : var[\tau] \vdash v := !v + 1 : comm$ is:

O: *run*
P: $read_v$ (what is the value of v)
O: 2 (O supplies the value 2)
P: $write(3)_v$ (write 3 into v)
O: ok_v (the assignment is complete)
P: *done*

In these plays O is not constrained to play a good variable for v, i.e. if in some next move P is trying to read the value of the variable v, O could answer with any value, not only with 3 as one would expect. But in case when the variable v is local, v is guaranteed to exhibit "good variable" behaviour and all moves pertaining to v are not an observable part of the term behaviour, i.e. are hidden.

3.2 CSP

CSP (Communicating Sequential Processes) is a language for modelling interacting components. Each component is specified through its behaviour which is given as a process. This section only introduces the CSP notation and the ideas used in this paper. For a fuller introduction to the language the reader is referred to [16].

CSP processes are defined in terms of the events that they can perform. The set of all possible events is denoted Σ. Events may be atomic in structure or may consist of a number of distinct components. For example, an event *write*.1

consists of two parts: a channel name *write*, and a data value 1. If N is a set of values that can be communicated down the channel *write*, then $write.N$ will be the set of events $\{write.n \mid n \in N\}$. Given a channel c, we can define the set of all events that can arise on the channel c, by $\{| c |\} = \{c.w \in \Sigma\}$.

We use the following collection of process operators:

$$P ::= p \mid STOP \mid SKIP \mid RUN_A \mid ?x : A \rightarrow P \mid \mu p.P \mid P_1 \Box P_2 \\ \mid P_1 \nleftarrow b \nrightarrow P_2 \mid P_1 \underset{A}{\|} P_2 \mid P \setminus A \mid P\,[a/b] \mid P_1 \fatsemi P_2$$

where A represents a set of events, P a process expression and p is a process name (or identifier).

The process $STOP$ performs no actions and never communicates. It is useful for providing a simple model of a deadlocked system. $SKIP$ is a process that successfully terminates causing the special event $\checkmark$ ($\checkmark$ is not in Σ). RUN_A can always communicate any event of set A desired by the environment. A choice process, $?x : A \rightarrow P$, can perform any event from set A and then behaves as P. For example, $RUN_A = ?x : A \rightarrow RUN_A$. Process $\mu p.P$, where P is any process involving p, represents recursion. It can return to its initial state and in that way communicate forever. The process $P_1 \Box P_2$ can behave either as P_1 or as P_2, its possible communications are those of P_1 and those of P_2. Conditional choice process $P_1 \nleftarrow b \nrightarrow P_2$ means the same as *if b then P else Q*, and has the obvious meaning as in all programming languages. Process $P_1 \underset{A}{\|} P_2$ runs P_1 and P_2 in parallel, making them synchronise on events in A and allowing all others events freely. The parallel combination terminates successfully when both component processes do it. This is known as distributed termination. Process $P \setminus A$, behaves as P except that events from A become invisible events τ (τ is not in Σ). The renaming operator [] is used to rename some events of a given process. We will only use injective renaming and notation $P[a/b]$ to mean that the event or channel b in P is replaced by a, and all others remain the same. Sequential composition $P_1 \fatsemi P_2$ runs P_1 until it terminates successfully producing the special event $\checkmark$, and then runs P_2.

CSP processes can be given semantics by sets of their traces. A trace is a finite sequence of events. A sequence *tr* is a trace of a process P if there is some execution of P in which exactly that sequence of events is performed. Examples of traces include $\langle\rangle$ (the empty trace, which is possible for any process) and $\langle a_1, a_2\rangle$ which is a possible trace of RUN_A, if $a_1, a_2 \in A$. The set $traces(P)$ is the set of all possible traces of process P.

Useful operators on traces are concatenation, $s^\frown t$ which constructs a trace from a pair of traces s and t by simply putting them together in this order, and restriction, if A is a set of events then the trace $tr \upharpoonright A$ is the trace *tr* restricted to events in the set A.

Using traces sets, we can define traces refinement. A process P_2 is a traces refinement of another, P_1, if all the possible sequences of communications which P_2 can do are also possible for P_1. Or more formally:

$$P_1 \sqsubseteq_T P_2 \;\Leftrightarrow\; traces(P_2) \subseteq traces(P_1) \tag{1}$$

CSP processes can also be described by transition systems or state machines. The transition system of a process is a directed graph showing the states which the process can go through and the events from $\Sigma \cup \{\checkmark, \tau\}$ that it can perform to get from one to another state. The successful termination $\checkmark$ is always the last event and leads to an end state Ω.

The FDR tool [6] is a refinement checker for CSP processes. It contains several procedures for compositional state-space reduction. Namely, before generating a transition system for a composite process, transition systems of its component processes can be reduced, while preserving semantics of the composite process. FDR is also optimised for checking refinements by processes which consist of a number of component processes composed by operators such as renaming, parallel composition and hiding.

4 CSP representation of game semantics

With each type θ, we associate a set of possible events - alphabet $\mathcal{A}_\theta$. The alphabet of a type contains events $\mathbf{q} \in \mathbb{Q}_\theta$ called questions, which are appended to channel with name Q, and for each question $\mathbf{q}$, there is a set of events $\mathbf{a} \in \mathbb{A}_\theta^{\mathbf{q}}$ called answers, which are appended to channel with name A.

$\mathcal{A}_{int} = \{0, \cdots, N_{max} - 1\}, \quad \mathcal{A}_{bool} = \{true, false\}$
$\mathbb{Q}_{exp[\tau]} = \{q\}, \quad \mathbb{A}^{q}_{exp[\tau]} = \mathcal{A}_\tau$
$\mathbb{Q}_{comm} = \{run\}, \quad \mathbb{A}^{run}_{comm} = \{done\}$
$\mathbb{Q}_{var[\tau]} = \{read, write.x \mid x \in \mathcal{A}_\tau\}, \quad \mathbb{A}^{read}_{var[\tau]} = \mathcal{A}_\tau, \quad \mathbb{A}^{write.x}_{var[\tau]} = \{ok\}$
$\mathbb{Q}_{\sigma_1 \times \cdots \times \sigma_k \to \sigma_0} = \{j.\mathbf{q} \mid \mathbf{q} \in \mathbb{Q}_{\sigma_j}, 0 \leq j \leq k\},$
$\mathbb{A}^{j.\mathbf{q}}_{\sigma_1 \times \cdots \times \sigma_k \to \sigma_0} = \{j.\mathbf{a} \mid \mathbf{a} \in \mathbb{A}^{\mathbf{q}}_{\sigma_j}\}, \text{for } 0 \leq j \leq k$
$\mathcal{A}_\theta = Q.\mathbb{Q}_\theta \cup A.\bigcup_{\mathbf{q} \in \mathbb{Q}_\theta} \mathbb{A}^{\mathbf{q}}_\theta$

A CSP representation is provided for typed terms-in-context $\Gamma \vdash M : \theta$, and it maps game semantics of each term to a process $[\![\Gamma \vdash M : \theta]\!]^{CSP}$. This process is defined over an alphabet $\mathcal{A}_{\Gamma \vdash M:\theta}$, defined as:

$\mathcal{A}_{x:\theta} = x.\mathcal{A}_\theta = \{x.\alpha \mid \alpha \in \mathcal{A}_\theta\}$
$\mathcal{A}_\Gamma = \bigcup_{x:\theta \in \Gamma} \mathcal{A}_{x:\theta}$
$\mathcal{A}_{\Gamma \vdash M:\theta} = \mathcal{A}_\Gamma \cup \mathcal{A}_\theta$

4.1 Expressions

$[\![\Gamma \vdash v : exp[\tau]]\!]^{CSP} = Q.q \to A.v \to SKIP, \; v \in \mathcal{A}_\tau$ is a constant
$[\![\Gamma \vdash not\ B : exp[bool]]\!]^{CSP} =$
$\quad [\![\Gamma \vdash B : exp[bool]]\!]^{CSP}[Q_1/Q, A_1/A_1] \underset{\{|Q_1, A_1|\}}{\parallel}$
$\quad (Q.q \to Q_1.q \to A_1?v : \mathcal{A}_{bool} \to A.(not\ b) \to SKIP) \setminus \{|\, Q_1, A_1 \,|\}$

$$
\begin{aligned}
&[\![\Gamma \vdash E_1 \bullet E_2 : exp[\tau]]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash E_1 : exp[\tau]]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_1,A_1|\}}{\parallel} \\
&\quad ([\![\Gamma \vdash E_2 : exp[\tau]]\!]^{CSP}[Q_2/Q, A_2/A] \underset{\{|Q_2,A_2|\}}{\parallel} \\
&\quad (Q.q \to Q_1.q \to A_1?v1 : \mathcal{A}_\tau \to Q_2.q \to A_2?v2 : \mathcal{A}_\tau \to \\
&\qquad A.(v1 \bullet v2) \to SKIP) \setminus \{|\ Q_2, A_2\ |\}) \setminus \{|\ Q_1, A_1\ |\}
\end{aligned}
$$

$$
[\![\Gamma \vdash \iota : exp[\tau]]\!]^{CSP} = Q.q \to \iota.Q.q \to \iota.A?v : \mathcal{A}_\tau \to A.v \to SKIP,\ \iota \in \Gamma
$$

Any constant $v : exp[\tau]$ is represented by a process which communicates the events a question q (what is the value of this expression), an answer v — the value of that constant, and then terminates successfully. The process that represents any arithmetic-logic operator $\bullet$ is defined in a compositional way, by parallel combination of the processes that represents both operands and a process that gets the operands values by synchronisation on the corresponding channels and returns the expected answer $(v1 \bullet v2)$. All events that participate in synchronisation are hidden. Arithmetic operators over a finite set of integers are interpreted as modulo some maximum value. The last process represents a free identifier ι of type $exp[\tau]$ in an obvious way.

4.2 Commands

$$
[\![\Gamma \vdash skip : comm]\!]^{CSP} = Q.run \to A.done \to SKIP
$$

$$
[\![\Gamma \vdash diverge : comm]\!]^{CSP} = STOP
$$

$$
\begin{aligned}
&[\![\Gamma \vdash C_1 \,;\, C_2 : comm]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash C_1 : comm]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_1,A_1|\}}{\parallel} \\
&\quad ([\![\Gamma \vdash C_2 : comm]\!]^{CSP}[Q_2/Q, A_2/A] \underset{\{|Q_2,A_2|\}}{\parallel} \\
&\quad (Q.run \to Q_1.run \to A_1.done \to Q_2.run \to A_2.done \to A.done \to SKIP) \\
&\qquad \setminus \{|\ Q_2, A_2\ |\}) \setminus \{|\ Q_1, A_1\ |\}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\Gamma \vdash C \,;\, E : exp[\tau]]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash C : comm]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_1,A_1|\}}{\parallel} \\
&\quad ([\![\Gamma \vdash E : exp[\tau]]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_2,A_2|\}}{\parallel} \\
&\quad (Q.q \to Q_1.run \to A_1.done \to Q_2.q \to A_2?v : \mathcal{A}_\tau \to A.v \to SKIP) \\
&\qquad \setminus \{|\ Q_2, A_2\ |\}) \setminus \{|\ Q_1, A_1\ |\}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\Gamma \vdash \mathit{if}\ B\ \mathit{then}\ C_1\ \mathit{else}\ C_2 : \mathit{comm}]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash B : \mathit{exp}[\mathit{bool}]]\!]^{CSP}[Q_0/Q, A_0/A] \underset{\{|Q_0,A_0|\}}{\parallel} \\
&\quad \Big(\big([\![\Gamma \vdash C_1 : \mathit{comm}]\!]^{CSP}[Q_1/Q, A_1/A] \,\Box\, (A.\mathit{done} \to \mathit{SKIP})\big) \underset{\{|Q_1,A_1,A|\}}{\parallel} \\
&\quad \Big(\big([\![\Gamma \vdash C_2 : \mathit{comm}]\!]^{CSP}[Q_2/Q, A_2/A] \,\Box\, (A.\mathit{done} \to \mathit{SKIP})\big) \underset{\{|Q_2,A_2,A|\}}{\parallel} \\
&\quad (Q.\mathit{run} \to Q_0.q \to A_0?v : \mathcal{A}_{bool} \to (Q_1.\mathit{run} \to A_1.\mathit{done} \to A.\mathit{done} \to \mathit{SKIP} \\
&\qquad \nless v \ngtr Q_2.\mathit{run} \to A_2.\mathit{done} \to A.\mathit{done} \to \mathit{SKIP})) \\
&\qquad\qquad\qquad\qquad \setminus\{|\, Q_2, A_2 \,|\}\Big)\setminus\{|\, Q_1, A_1 \,|\}\Big)\setminus\{|\, Q_0, A_0 \,|\}
\end{aligned}
$$

$$
\begin{aligned}
&[\![\Gamma \vdash \mathit{while}\ B\ \mathit{do}\ C : \mathit{comm}]\!]^{CSP} = \\
&\quad \mu\, p'.([\![B : \mathit{comm}]\!]^{CSP}[Q_1/Q, A_1/A] \,;\, p') \,\Box\, (A.\mathit{done} \to \mathit{SKIP})) \underset{\{|Q_1,A_1,A|\}}{\parallel} \\
&\quad (\mu\, p''.([\![C : \mathit{comm}]\!]^{CSP}[Q_2/Q, A_2/A] \,;\, p'') \,\Box\, (A.\mathit{done} \to \mathit{SKIP})) \underset{\{|Q_2,A_2,A|\}}{\parallel} \\
&\quad (Q.\mathit{run} \to \mu\, p.\Big(Q_1.q \to A_1?v : \mathcal{A}_{bool} \to (Q_2.\mathit{run} \to A_2.\mathit{done} \to p \nless v \ngtr \\
&\qquad A.\mathit{done} \to\ \mathit{SKIP})\Big)) \setminus \{|\, Q_2, A_2 \,|\}) \setminus \{|\, Q_1, A_1 \,|\}
\end{aligned}
$$

$$
[\![\Gamma \vdash \iota : \mathit{comm}]\!]^{CSP} = Q.\mathit{run} \to \iota.Q.\mathit{run} \to \iota.A.\mathit{done} \to A.\mathit{done} \to \mathit{SKIP},\ \iota \in \Gamma
$$

The command *skip* is represented by the corresponding question and answer events for commands followed by ✓ event. The command *diverge* is represented by the deadlocked process $STOP$, which matches with its game semantics, namely there is not any complete play for *diverge*. Composition of commands is represented as parallel combination of the commands processes and a process which starts this composition with $Q.run$ event, then by synchronisation executes the first and the second command, and in the end terminates successfully hiding all synchronisation events. Composition of a command with an expression and branching are represented in analogues way. Slightly more complicated is representation of loop, where we have parallel combination of three recursive processes. The first two can run the guard and the body process zero or more times, and the third one executes the body process and unwind as long as the value synchronised with the guard process is true. In the moment when this value will be *false*, the all three processes terminate successfully. The last process represents a free identifier ι of type command.

4.3 Variables

$$
\begin{aligned}
&[\![\Gamma \vdash V : \mathit{var}[\tau]]\!]^{CSP} = \\
&\quad \mu\, p.\big((Q.\mathit{read} \to V.Q.\mathit{read} \to V.A?v : \mathcal{A}_\tau \to A.v \to p) \,\Box \\
&\qquad (Q.\mathit{write}?v : \mathcal{A}_\tau \to V.Q.\mathit{write}.v \to V.A.\mathit{ok} \to A.\mathit{ok} \to p) \,\Box \\
&\qquad \mathit{SKIP}\big)
\end{aligned}
$$

The term V of type $var[\tau]$, is represented as external choice between a process for reading a value from the variable and a process for writing a value into it.

Representations of assignment command and de-referencing of a variable are:

$$
\begin{aligned}
&[\![\Gamma \vdash V := M : comm]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash M : exp[\tau]]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_1,A_1|\}}{\parallel} \\
&\quad \big([\![\Gamma \vdash V : var[\tau]]\!]^{CSP}[Q_2/Q, A_2/A] \underset{\{|Q_2,A_2|\}}{\parallel} \\
&\quad (Q.run \to Q_1.q \to A_1?v : \mathcal{A}_\tau \to Q_2.write.v \to A_2.ok \\
&\qquad \to A.done \to SKIP) \setminus \{|\ Q_2, A_2\ |\}\big) \setminus \{|\ Q_1, A_1\ |\} \\
&[\![\Gamma \vdash !V : exp[\tau]]\!]^{CSP} = \\
&\quad [\![\Gamma \vdash V : var[\tau]]\!]^{CSP}[Q_1/Q, A_1/A] \underset{\{|Q_1,A_1|\}}{\parallel} \\
&\quad (Q.q \to Q_1.read \to A_1?v : \mathcal{A}_\tau \to A.v \to SKIP) \setminus \{|\ Q_1, A_1\ |\}
\end{aligned}
$$

4.4 Local variables

The semantics of a local variable block consist of two operations, imposing the good variable behaviour on the local variable and removing all references to the variable, as it becomes invisible outside its binding scope. The first condition is accomplished with synchronising the process that represents the local-variable block with a process $U_{x:var[\tau],a_\tau}$ initialised with $a_{int} = 0$ or $a_{bool} = false$ and defined as:

$$
\begin{aligned}
U_{x:var[\tau],v:\tau} = &(\ x.Q.read \to x.A!v \to U_{x:var[\tau],v:\tau}\ ) \,\Box \\
&(\ x.Q.write?v' : \tau \to x.A.ok \to U_{x:var[\tau],v':\tau}\ ) \,\Box \\
&(\ A.done \to SKIP\ )
\end{aligned}
$$

The second condition is realised very easily, just by hiding all events that arise on the channel with the local-variable name. So, the CSP representation of the local-variable block is:

$$
\begin{aligned}
&[\![\Gamma \ \vdash\ new[\tau]\ x\ in\ M : comm]\!]^{CSP} = \\
&\qquad \big([\![\Gamma \ \vdash\ M : comm]\!]^{CSP} \underset{\{|x,A|\}}{\parallel} U_{x:var[\tau],a_\tau}\big) \setminus \{|\ x\ |\}
\end{aligned}
$$

4.5 Application and functions

Application $\iota(M_1 \ldots M_k)$, where ι is a free identifier, is represented by a process that communicates the usual question and answer events for the return type of the function on the channel $\iota.0$, and runs zero or more times any processes of the function arguments. Arguments events are appended on the channels comprised of the function name followed by the index of that argument.

$$
\begin{aligned}
&[\![\Gamma \vdash \iota(M_1 \ldots M_k) : \sigma]\!]^{CSP} = Q?\mathbf{q} : \mathbb{Q}_\sigma \to \iota.0.Q.\mathbf{q} \to \\
&\quad \mu L.\Big(\Big(\Box_{j=1,k}\big([\![\Gamma \vdash M_j : \sigma_j]\!]^{CSP} \underset{\{|Q,A|\}}{\parallel} (\iota.j.Q?\mathbf{q} : \mathbb{Q}_{\sigma_j} \to Q.\mathbf{q} \to A?\mathbf{a} : \mathbb{A}^{\mathbf{q}}_{\sigma_j} \\
&\quad \to \iota.j.A.a \to L)\big) \setminus \{|\ Q, A\ |\}\Big) \,\Box\, SKIP\Big) \,;\, \iota.0.A?\mathbf{a} : \mathbb{A}^{\mathbf{q}}_{\sigma} \to A.\mathbf{a} \to SKIP
\end{aligned}
$$

Finally, we need to give CSP representation of the *let* construct.
$[\![\Gamma \vdash let\ \iota(\iota_1 : \sigma_1, \ldots, \iota_k\ : \sigma_k)\ =\ N\ in\ M : \sigma)]\!]^{CSP} =$
$\big([\![\Gamma \vdash M : \sigma]\!]^{CSP} \underset{\{|\iota|\}}{\|}$
$[\![\Gamma \vdash N : \sigma']\!]^{CSP}[\iota.0.Q/Q, \iota.1/\iota_1, \ldots, \iota.k/\iota_k, \iota.0.A/A]\big) \setminus \{|\ \iota\ |\}$

CSP process of *let* construct is simply the CSP process of M, where a subprocess that represents call to a function $\iota(M_1, \cdots, M_k)$ is synchronised with the process $[\![\Gamma \vdash N : \sigma']\!]^{CSP}$ with appropriately renamed events.

5 Correctness and decidability

Our first result is that, for any term, the set of all terminated traces of its CSP interpretation is isomorphic to the language of its regular language interpretation, as defined in [7].

Theorem 1. *For any term $\Gamma \vdash M : \theta$, we have:*

$$\mathcal{L}_R(\Gamma \vdash M : \theta) \ \overset{\phi}{\cong}\ \mathcal{L}_{CSP}(\Gamma \vdash M : \theta) \tag{2}$$

where

$$\mathcal{L}_R(\Gamma \vdash M : \theta) = \mathcal{L}([\![\Gamma \vdash M : \theta]\!]^R) \qquad \textit{(defined in [7])} \tag{3}$$

$$\mathcal{L}_{CSP}(\Gamma \vdash M : \theta) = \{tr \ \mid\ tr^\frown\langle\checkmark\rangle \in traces([\![\Gamma \vdash M : \theta]\!]^{CSP})\} \tag{4}$$

and ϕ is defined by:

$$\begin{array}{l} \phi(\langle a_1, \ldots, a_k\rangle) = \phi(a_1) \cdot \ldots \cdot \phi(a_k) \\ \phi(Q.m) = m \qquad \phi(A.n) = n \\ \phi(x.Q.m) = m^x\ \phi(x.A.n) = n^x \end{array}$$

Proof. The proof of this Theorem is by induction on the typing rules defined in Table 1.

Consider the basic term $\Gamma \vdash v : exp[\tau]$, where v is a constant. It holds that:
$\mathcal{L}_R(\Gamma \vdash v : exp[\tau]) = \{q \cdot v\}$, and
$traces([\![\Gamma \vdash v : exp[\tau]]\!]^{CSP}) = \{\langle Q.q\rangle, \langle Q.q, A.v\rangle, \langle Q.q, A.v, \checkmark\rangle\}$.
So, $\mathcal{L}_{CSP}(\Gamma \vdash v : exp[\tau]) = \{\langle Q.q, A.v\rangle\} \cong \{q \cdot v\} = \mathcal{L}_R(\Gamma \vdash v : exp[\tau])$.

The proofs for the other base terms, commands *skip* and *diverge* go in a similar manner.

Consider a term $\Gamma \vdash not\ B : exp[bool]$, assuming that the claim holds for its immediate constituents i.e. $\mathcal{L}_R(\Gamma \vdash B : exp[bool]) \cong \mathcal{L}_{CSP}(\Gamma \vdash B : exp[bool])$.
$\mathcal{L}_R(\Gamma \vdash not\ B) = \sum_{v \in \mathcal{A}_{bool}} q \cdot R^v_{not\ B} \cdot v = \sum_{v \in \mathcal{A}_{bool}} q \cdot R^v_B \cdot (not\ v) =$
$= \ q \cdot w'_B \cdot (not\ v)$, where $w_B = q \cdot w'_B \cdot v \in \mathcal{L}_R(\Gamma \vdash B)$
From inductive hypotheses, for each word $w_B \in \mathcal{L}_R(\Gamma \vdash B)$, there is a successful trace $tr_B \in \mathcal{L}_{CSP}(\Gamma \vdash B)$, such that:
$tr_B = \langle Q.q\rangle^\frown tr'^\frown_B\langle A.v\rangle \cong q \cdot w'_B \cdot v = w_B$, where the value of v is the same on the both sides. Now, from the definition of $[\![\Gamma \vdash not\, B]\!]^{CSP}$, parallel operator

and hiding, we have:
$\mathcal{L}_{CSP}(\Gamma \vdash not\, B) =$
$$\{\langle Q.q\rangle^\frown tr'_B{}^\frown\langle A.(not\ v)\rangle \mid tr_B = \langle Q.q\rangle^\frown tr'_B{}^\frown\langle A.v\rangle \in \mathcal{L}_{CSP}(\Gamma \vdash B)\} \cong$$
$$\{q \cdot w'_B \cdot (not\, v) \mid w_B = q \cdot w'_B \cdot v \in \mathcal{L}_R(\Gamma \vdash B)\} = \mathcal{L}_R(\Gamma \vdash not\, B)$$

Analogously, we can prove the claim for arithmetic/logic operator terms and equality terms.

For a term $\Gamma \vdash while\ B\ do\ C : comm$, we have:
$\mathcal{L}_R(\Gamma \vdash while\, B\, do\, C) = run \cdot R_{while\ B\ do\ C} \cdot done =$
$run \cdot (R_B^{true} \cdot R_C)^* \cdot R_B^{false} \cdot done =$
$$\begin{cases} run \cdot w_B^0 \cdot done & \text{if } q \cdot w_B^0 \cdot false \in \mathcal{L}_R(B)^0 \\ run \cdot w_B^0 \cdot w_C \cdot w_B^1 \cdot done & \text{if } q \cdot w_B^1 \cdot false \in \mathcal{L}_R(B)^1,\, q \cdot w_B^0 \cdot true \in \mathcal{L}_R(B)^0 \\ \quad \dots & \\ run \cdot w_B^0 \cdot \dots \cdot w_C \cdot w_B^n \cdot done & \text{if } q \cdot w_B^n \cdot false \in \mathcal{L}_R(B)^n,\, q \cdot w_B^i \cdot true \in \mathcal{L}_R(B)^i \end{cases}$$
where, w_C represents a word from $\mathcal{L}_R(C)$ and $\mathcal{L}_R(B)^n$ is a regular language that represents the guard B after $n-$times executing of command C. From the definition of $[\![\Gamma \vdash while\, B\, do\, C]\!]^{CSP}$ and operators that appear there, follows:
$\mathcal{L}_{CSP}(\Gamma \vdash while\, B\, do\, C) =$
$$\begin{cases} \langle Q.run\rangle^\frown tr_B^0{}^\frown\langle A.done\rangle & \text{if } \langle Q.q\rangle^\frown tr_B^0{}^\frown\langle A.false\rangle \in \mathcal{L}_{CSP}(B)^0 \\ \langle Q.run\rangle^\frown tr_B^0{}^\frown tr_C{}^\frown tr_B^1\langle A.done\rangle & \text{if } \langle Q.q\rangle^\frown tr_B^1{}^\frown\langle A.false\rangle \in \mathcal{L}_{CSP}(B)^1 \\ & \quad \langle Q.q\rangle^\frown tr_B^0{}^\frown\langle A.true\rangle \in \mathcal{L}_{CSP}(B)^0 \\ \quad \dots & \\ \langle Q.run\rangle^\frown tr_B^0{}^\frown \dots {}^\frown tr_C{}^\frown tr_B^n\langle A.done\rangle & \text{if } \langle Q.q\rangle^\frown tr_B^n{}^\frown\langle A.false\rangle \in \mathcal{L}_{CSP}(B)^n \\ & \quad \langle Q.q\rangle^\frown tr_B^i{}^\frown\langle A.true\rangle \in \mathcal{L}_{CSP}(B)^i,\, i < n \end{cases}$$
where, tr^C represents a trace from $\mathcal{L}_{CSP}(C)$ and $\mathcal{L}_{CSP}(B)^n$ is a set of traces of the guard B after $n-$times unwinding the recursion p.

From inductive hypotheses, we have $\mathcal{L}_R(C) \cong \mathcal{L}_{CSP}(C)$ and $\mathcal{L}_R(B)^0 \cong \mathcal{L}_{CSP}(B)^0$, but it also holds that $\mathcal{L}_R(B)^n \cong \mathcal{L}_{CSP}(B)^n$ because they are obtained in the same manner, after $n-$times executing the command C, which is isomorphic in both representations. So, follows that for any loop, no matter how many times it will be unwind, we will get a trace in $\mathcal{L}_{CSP}(while\, B\, do\, C)$ that is isomorphic to a word in $\mathcal{L}_R(while\, B\, do\, C)$. This implies that these two sets are isomorphic.

The proofs for other commands are similar.

The definitions of local-variable block and application are analogous in both representations.

Regular-language representation of *let* construct

$$\Gamma \vdash let\, \iota\, (\iota_1 : \sigma_1, \dots, \iota_k : \sigma_k) = N\ in\ M : \sigma$$

in [7] is based on an environment u which maps identifiers to regular languages, $u : dom\Gamma \to R_{\mathcal{A}}$, i.e.

$$\begin{aligned} &[\![let\, \iota\, (\iota_1 : \sigma_1, \dots, \iota_k : \sigma_k) = N\ in\ M : \sigma]\!]^R = \\ &\quad [\![M : \sigma]\!]^R u\, [\, \iota \to [\![N : \sigma']\!]^R u\, [\, \iota_1 \to [\![\iota_1 : \sigma_1]\!]^R, \dots, \iota_k \to [\![\iota_k : \sigma_k]\!]^R\,]\,] \end{aligned} \tag{5}$$

The corresponding definition of functions without using environments is:

$$\begin{aligned} &[\![let\, \iota\, (\iota_1 : \sigma_1, \ldots, \iota_k : \sigma_k) = N\, in\, M : \sigma]\!]^R = \\ &\quad [\![M : \sigma]\!]^R \Big[[\![N : \sigma']\!]^R \left[\mid M_j : \sigma_j \mid_v / \mathbf{q}^{\iota j} \cdot v^{\iota j} \right] \Big/ \\ &\qquad\qquad \mathbf{q}^{\iota} \cdot \left(\textstyle\sum_{1 \leqslant j \leqslant k} \sum_v \mathbf{q}^{\bullet \iota j} \cdot \mid M_j : \sigma_j \mid_v \cdot v^{\bullet \iota j} \right)^* \cdot \mathbf{a}^{\iota} \Big] \end{aligned} \tag{6}$$

where $\mathbf{q}\cdot \mid M_j : \sigma_j \mid_v \cdot v$ is a word with answer v from $[\![M_j : \sigma_j]\!]^R$.

Suppose $w \in \mathcal{L}_R(let\, \iota\, (\iota_1 : \sigma_1, \ldots, \iota_k : \sigma_k) = N\, in\, M : \sigma)$.

If w represents a complete play of construct *let* without calls to the function ι, then w is a word from $\mathcal{L}_R(M)$ and from inductive hypotheses $\mathcal{L}_R(M) \cong \mathcal{L}_{CSP}(M)$, follows that there exists a trace $tr \in \mathcal{L}_R(let\, \iota\, (\iota_1 : \sigma_1, \ldots, \iota_k : \sigma_k) = N\, in\, M : \sigma)$ that is isomorphic to w.

If w represents a *let* construct with a call to the function ι, then to prove that there exists an isomorphic successful trace tr in the *let* process, it will be enough to prove that subword w^{ι} which represents call to ι is isomorphic with a corresponding subtrace tr^{ι} in tr.

From the definition (6), we have:
$w^{\iota} = w^N[\mid M_j : \sigma_j \mid_v / \mathbf{q}^{\iota j} \cdot v^{\iota j}],\ w^N \in \mathcal{L}_R(N),\ \mathbf{q}\cdot \mid M_j : \sigma_j \mid_v \cdot v \in \mathcal{L}_R(M_j)$
From the CSP representation of the *let* construct, we have:
$tr^{\iota} = tr^N[tr_j^v / \langle \iota.j.Q.\mathbf{q}, \iota.j.A.v \rangle],\ tr^N \in \mathcal{L}_{CSP}(N),\ \langle Q.\mathbf{q} \rangle ^\frown tr_j^v {}^\frown \langle A.v \rangle \in \mathcal{L}_{CSP}(M_j)$
Since $\mathcal{L}_R(N) \cong \mathcal{L}_{CSP}(N)$ and $\mathcal{L}_R(M_j) \cong \mathcal{L}_{CSP}(M_j)$ for $j = 1, \ldots, k$, it holds that $w^N \cong tr^N$ and $\mid M_j : \sigma_j \mid_v \cong tr_j^v$, for $j = 1, \ldots, k$, and this implies $w^{\iota} \cong tr^{\iota}$.
$\square$

Two terms M and N in type context Γ and of type θ are observationally equivalent, written $\Gamma \vdash M \equiv_\theta N$, iff for any term-with-hole $C[-]$ such that both $C[M]$ and $C[N]$ are closed terms of type *comm*, $C[M]$ converges iff $C[N]$ converges. It was proved in [1] that this coincides to equality of sets of complete plays of the strategies for M and N, i.e. that the games model is fully abstract. (Operational semantics of IA, and a definition of convergence for terms of type *comm* in particular, can be found in the same paper.)

For the IA fragment treated in this paper, it was shown in [7] that observational equivalence coincides with equality of regular language interpretations. By Theorem 1, we have that observational equivalence corresponds to two traces refinements:

Corollary 1 (Observational equivalence).

$$\begin{aligned} \Gamma \vdash M \equiv_\theta N \ \Leftrightarrow\ & [\![\Gamma \vdash M : \theta]\!]^{CSP} \,\square\, RUN_\Sigma \ \sqsubseteq_T\ [\![\Gamma \vdash N : \sigma]\!]^{CSP} \ \wedge \\ & [\![\Gamma \vdash N : \theta]\!]^{CSP} \,\square\, RUN_\Sigma \ \sqsubseteq_T\ [\![\Gamma \vdash M : \sigma]\!]^{CSP} \end{aligned} \tag{7}$$

Proof.

$$\begin{aligned} \Gamma \vdash M \equiv_\theta N \ &\overset{(i)}{\Leftrightarrow} \mathcal{L}([\![\Gamma \vdash M : \sigma]\!]^R) \ = \ \mathcal{L}([\![\Gamma \vdash N : \theta]\!]^R) \\ &\overset{(ii)}{\Leftrightarrow} \mathcal{L}_{CSP}(\Gamma \vdash M : \theta) \ = \ \mathcal{L}_{CSP}(\Gamma \vdash N : \theta) \\ &\overset{(iii)}{\Leftrightarrow} [\![\Gamma \vdash M : \theta]\!]^{CSP} \,\square\, RUN_\Sigma \ \sqsubseteq_T\ [\![\Gamma \vdash N : \theta]\!]^{CSP} \ \wedge \\ &\qquad [\![\Gamma \vdash N : \theta]\!]^{CSP} \,\square\, RUN_\Sigma \ \sqsubseteq_T\ [\![\Gamma \vdash M : \theta]\!]^{CSP} \end{aligned}$$

where (i) is shown in [7], and (ii) holds by Theorem 1. We will prove (iii) by proving the following two equivalences:

$$\mathcal{L}_{CSP}(M:\theta) \subseteq \mathcal{L}_{CSP}(N:\theta) \;\Leftrightarrow\; [\![N:\theta]\!]^{CSP} \;\square\; RUN_{\Sigma} \;\sqsubseteq_T\; [\![M:\theta]\!]^{CSP}$$
$$\mathcal{L}_{CSP}(N:\theta) \subseteq \mathcal{L}_{CSP}(M:\theta) \;\Leftrightarrow\; [\![M:\theta]\!]^{CSP} \;\square\; RUN_{\Sigma} \;\sqsubseteq_T\; [\![N:\theta]\!]^{CSP}$$

For the first equivalence, suppose $\mathcal{L}_{CSP}(\Gamma \vdash M:\theta) \subseteq \mathcal{L}_{CSP}(\Gamma \vdash N:\theta)$ $(*)$. If there is some trace in $[\![\Gamma \vdash M]\!]^{CSP}$ that finishes successfully with event $\langle\checkmark\rangle$ then from assumption $(*)$, this trace is also in $[\![\Gamma \vdash N:\theta]\!]^{CSP}$. All other traces in $[\![\Gamma \vdash M]\!]^{CSP}$ that do not finish with $\langle\checkmark\rangle$, are certainly in traces set of RUN_{Σ}. So, $traces([\![\Gamma \vdash M]\!]^{CSP}) \subseteq traces([\![\Gamma \vdash N]\!]^{CSP}) \cup traces(RUN_{\Sigma})$ which implies that the first direction holds.

Conversely, suppose $[\![\Gamma \vdash N:\theta]\!]^{CSP} \;\square\; RUN_{\Sigma} \;\sqsubseteq_T\; [\![\Gamma \vdash M:\theta]\!]^{CSP}$ $(**)$. Let $tr \in \mathcal{L}_{CSP}(\Gamma \vdash M:\theta)$, which means that $tr^\frown\langle\checkmark\rangle \in traces([\![\Gamma \vdash M:\theta]\!]^{CSP})$. This trace $tr^\frown\langle\checkmark\rangle$ is not certainly in $traces(RUN_{\Sigma})$ because $\checkmark \notin \Sigma$, so it must $tr^\frown\langle\checkmark\rangle \in traces([\![\Gamma \vdash N:\theta]\!]^{CSP})$, i.e. $tr \in \mathcal{L}_{CSP}(\Gamma \vdash N:\theta)$, which implies that $\mathcal{L}_{CSP}(\Gamma \vdash M:\theta) \subseteq \mathcal{L}_{CSP}(\Gamma \vdash N:\theta)$.

The proof for the other equivalence is analogous. □

Refinement checking in FDR terminates for finite-state processes, i.e. those whose transition systems are finite. Our next result confirms that this is the case for the processes interpreting the IA terms. As a corollary, we have that observational equivalence is decidable using FDR.

Theorem 2. *For any term $\Gamma \vdash M:\theta$, the CSP process $[\![\Gamma \vdash M:\theta]\!]^{CSP}$ is finite state.*

Proof. This follows by induction on typing rules, using the fact that a process can have infinitely many states only if it uses either a choice operator $?x : A \to P$, where A is an infinite set and P varying with x, or certain kinds of recursion.

Consider the loop process, in particular the third process in the parallel composition, which is recursive,
$Q.run \to \mu\, p.(Q_1.q \to A_1?v : \mathcal{A}_{bool} \to$
$(Q_2.run \to A_2.done \to p \not< v \not> A.done \to \; SKIP))$
It is easy to check that this process is finite state, only with exploring the all possible states that could be reached from initial one. The transition system of this process is shown on Figure 1.

In the same manner, we can also show that the other recursive processes used in representation of some terms, are finite state. □

Corollary 2 (Decidability). *Observational equivalence between terms of second-order recursion-free IA with iteration and finite data types is decidable by two traces refinements between finite-state CSP processes.* □

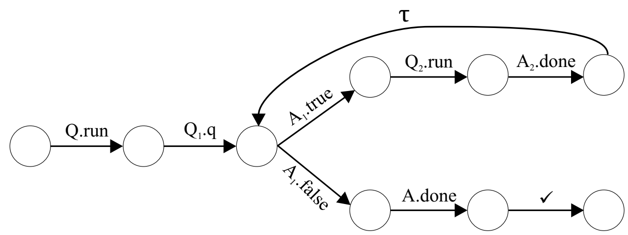

Fig. 1. A transition system

5.1 Example equivalences

We now consider several example equivalences and prove them using the CSP model.

Example 1. $\Gamma \vdash \mathit{while\ true\ do\ C} \ \equiv_{comm} \ \mathit{diverge}$.

We prove the first traces refinement of the equivalence (7), i.e.
$[\![\Gamma \vdash \mathit{while\ true\ do\ C} : \mathit{comm}]\!]^{CSP} \,\Box\, RUN_{\Sigma} \sqsubseteq_T [\![\Gamma \vdash \mathit{diverge} : \mathit{comm}]\!]^{CSP} \quad (*)$
Since $[\![\Gamma \vdash \mathit{diverge} : \mathit{comm}]\!]^{CSP} = STOP$, and the process $STOP$ traces refines any other process, it implies that $(*)$ holds.

The second traces refinement is:
$[\![\Gamma \vdash \mathit{diverge} : \mathit{comm}]\!]^{CSP} \,\Box\, RUN_{\Sigma} \sqsubseteq_T [\![\Gamma \vdash \mathit{while\ true\ do\ C} : \mathit{comm}]\!]^{CSP} \quad (**)$
We have:

$$\begin{aligned}
&\mathit{traces}([\![\Gamma \vdash \mathit{while\ true\ do\ C} : \mathit{comm}]\!]^{CSP}) = \\
&\quad = \{\langle\rangle, \langle Q.run\rangle, \langle Q.run\rangle^\frown tr', \langle Q.run\rangle^\frown tr, \langle Q.run\rangle^\frown tr^\frown tr', \ldots \mid \\
&\qquad \langle Q.run\rangle^\frown tr^\frown \langle A.done\rangle^\frown \langle\checkmark\rangle \in \mathit{traces}([\![\Gamma \vdash C]\!]^{CSP}), tr' \leqslant tr\}
\end{aligned}$$

Since there is not any sequence with successful termination in the traces set of $[\![\Gamma \vdash \mathit{while\ true\ do\ C} : \mathit{comm}]\!]^{CSP}$, all sequences from this traces set will be also in the traces set of RUN_{Σ} process, so $(**)$ holds too. □

Example 2. $C : \mathit{comm} \vdash \mathit{new}[\tau]\, x \ \mathit{in}\ C : \mathit{comm} \ \equiv_{comm} \ C : \mathit{comm}$.

This simple equivalence reflects the fact that a non-locally defined command cannot modify a local variable [7].

$$\begin{aligned}
&[\![C : \mathit{comm} \vdash \mathit{new}[\tau]\, x \ \mathit{in}\ C : \mathit{comm}]\!]^{CSP} = \\
&\ [\![C : \mathit{comm}, x : \mathit{var}[\tau] \vdash C : \mathit{comm}]\!]^{CSP} \underset{|Q,A|}{\parallel} (Q.run \to U(x, a_\tau)) \setminus \{\mid x \mid\} = \\
&\ (Q.run \to C.Q.run \to C.A.done \to A.done \to SKIP) \underset{|Q,A|}{\parallel} (Q.run \to U(x, a_\tau)) \\
&\qquad \setminus \{\mid x \mid\} = \\
&\ Q.run \to C.Q.run \to C.A.done \to A.done \to SKIP \ = \\
&\ [\![C : \mathit{comm} \vdash C : \mathit{comm}]\!]^{CSP}
\end{aligned}$$

The CSP processes that represent both sides are equal, so their traces sets are equal too. This implies that both traces refinements (7) hold. □

Example 3.

$$\begin{array}{ll} M : comm \to comm \vdash & \\ new[int]\ x\ in & \\ \ x := 0 \mathbin{;} M(x := 1) \mathbin{;} & \equiv_{comm} M(diverge) \\ \ if\ !x = 1\ then\ diverge\ else\ skip & \end{array}$$

This example captures the intuition that changes to the state are irreversible [7].

We proceed evaluations in a bottom-up fashion.

$[\![M : comm \to comm, x : var[int] \vdash M(v := 1) : comm]\!]^{CSP} =$
$(Q.run \to M.0.Q.run \to \mu L.\big(([\![x := 1]\!]^{CSP} \underset{|Q,A|}{\|} (M.1.Q.run \to Q.run \to$
$A.done \to M.1.A.done \to L)\backslash\{|\ Q, A\ |\})\Box SKIP\big) \mathbin{;} M.0.A.done \to$
$A.done \to SKIP =$
$Q.run \to M.0.Q.run \to \mu L.\big((M.1.Q.run \to x.Q.write.1 \to x.A.ok \to$
$M.1.A.done \to L) \Box SKIP\big) \mathbin{;} M.0.A.done \to A.done \to SKIP$

$[\![x : var[int] \vdash if\ !x = 1\ then\ diverge\ else\ skip : comm]\!]^{CSP} =$
$Q.run \to x.Q.read \to x.A?v : \mathcal{A}_{int} \to (STOP \not< v = 1 \not> A.done \to SKIP)$

Using these two processes, for the left-hand side process we have:

$[\![LHS]\!]^{CSP} =$
$\big(Q.run \to x.Q.wrire.0 \to x.A.ok \to M.0.Q.run \to$
$\mu L.\big((M.1.Q.run \to x.Q.write.1 \to x.A.ok \to M.1.A.done \to L) \Box SKIP\big) \mathbin{;}$
$M.0.A.done \to x.Q.read \to x.A?v : \mathcal{A}_{int} \to (STOP \not< v = 1 \not> A.done \to$
$SKIP)\big) \underset{|Q,A,x|}{\|} (Q.run \to U(x,0)) \setminus \{|\ x\ |\}$

If we only consider traces with successful termination for this process,
$\mathcal{L}_{CSP}(LHS) = \{\langle Q.run, M.0.Q.run, M.0.A.done, A.done\rangle\}$

For the right-hand side process, it holds that:

$[\![M : comm \to comm \vdash M(diverge) : comm]\!]^{CSP} =$
$Q.run \to M.0.Q.read \to \mu L.(STOP \Box SKIP) \mathbin{;} M.0.A.done \to A.done \to SKIP$

So, $\mathcal{L}_{CSP}(LHS) = \mathcal{L}_{CSP}(RHS)$, and according to Corollary 1, these two terms are equivalent. □

5.2 Property verification

Suppose ϕ is any property of terms with type context Γ and type θ such that the set of all behaviours which satisfy ϕ is a regular language $\mathcal{L}(\phi)$ over $\mathcal{A}_\Gamma \cup \mathcal{A}_\theta$. A finite-state CSP process whose set of terminated traces equals $\mathcal{L}(\phi)$ can then be constructed. Thus, we can verify whether a term $\Gamma \vdash M : \theta$ satisfies ϕ by checking the traces refinement:

$$P_\phi \Box RUN_\Sigma \sqsubseteq_T [\![\Gamma \vdash M : \theta]\!]^{CSP} \qquad (8)$$

Example 4. Consider a term $x : var[\tau], c : comm \vdash M : comm$. We want to check the property "a value is written into x before c is called". A CSP process which interprets this property is:

$$P_\phi = \mu\, p.\Big((?e : \mathcal{A}_{comm} \cup \mathcal{A}_{x:var[\tau] \setminus \{|x.Q.write|\}} \rightarrow p) \,\square\, \big(x.Q.write?v : \mathcal{A}_\tau \rightarrow \mu\, p'.(?e : \mathcal{A}_{comm} \cup \mathcal{A}_{x:var[\tau]} \rightarrow p') \,\square\, (c.Q.run \rightarrow \mu\, p''.(?e : \mathcal{A}_{\Gamma \vdash M:comm} \rightarrow p'') \,\square\, SKIP)\big)\Big) \,\square$$

6 Experimental results

We have implemented a compiler from IA terms-in-context into CSP processes which represent their game semantics. The input to the compiler is code, with some simple type annotations to indicate what finite sets of integers will be used to model integer variables.

Here, we discuss the modelling of a sorting program, report the results from our tool and compare them with the tool based on regular expressions [9]. We will analyse the bubble-sort algorithm, whose implementation is given in Figure 2. The code includes a meta variable n, representing array size, which will be replaced by several different values. The integers stored in the array are of type $int\%3$, i.e. 3 distinct values 0, 1 and 2, and the type of index i is $int\%n{+}1$, i.e. one more than the size of the array. The program first copies the input array $x[\,]$ into a local array $a[\,]$, which is then sorted and copied back into $x[\,]$. The array being effectively sorted, $a[\,]$, is not visible from the outside of the program because it is locally defined, and only reads and writes of the non-local array $x[\,]$ are seen in the model. The transition system of final (compressed) model process for $n = 2$ is shown in Figure 3. It illustrates the dynamic behaviour of the program, where the left-side half of the model reads all possible combinations of values from $x[\,]$, while the right-side half writes out the same values, but in sorted order.

Table 2 contains the experimental results for model generation. We ran FDR on a Research Machines Xeon with 2GB RAM. The results from the tool based on regular expressions were obtained on a SunBlade 100 with 2GB RAM [9]. We list the execution time, the size of the largest generated state machine during model generation, and the size of the final compressed model. In the CSP approach, the process output by our compiler was input into FDR, which was instructed to generate a transition system for it by applying a number of compositional state-space reduction algorithms. The results confirm that both approaches give isomorphic models, where the CSP models have an extra state due to representing termination by a ✓ event.

We expect FDR to perform even better in property verification. For checking refinement by a composite process, FDR does not need to generate an explicit model of it, but only models of its component processes. A model of the composite process is then generated on-the-fly, and its size is not limited by available RAM, but by disk size.

$$
\begin{array}{l}
\mathit{var\ int}\%_3\ x[n];\ \ \vdash \\
\mathit{new\ int}\%_3\ a[n]\ \mathit{in} \\
\mathit{new\ int}\%_{n+1}\ i\ \mathit{in} \\
\mathit{while}\ (i < n)\ \ \{a[i] := x[i];\ \ \ i := i + 1;\ \} \\
\mathit{new\ boolean\ flag} := \mathit{true\ in} \\
\mathit{while}(\mathit{flag})\{ \\
\quad i := 0; \\
\quad \mathit{flag} := \mathit{false}; \\
\quad \mathit{while}\ (i < n - 1)\ \ \{ \\
\quad\quad \mathit{if}\ (a[i] > a[i+1])\ \ \{ \\
\quad\quad\quad \mathit{flag} := \mathit{true}; \\
\quad\quad\quad \mathit{new\ int}\%_3\ \mathit{temp\ in} \\
\quad\quad\quad \mathit{temp} := a[i]; \\
\quad\quad\quad a[i] := a[i+1]; \\
\quad\quad\quad a[i+1] := \mathit{temp};\ \ \ \} \\
\quad\quad i := i + 1;\ \ \ \}\ \} \\
i := 0; \\
\mathit{while}\ (i < n)\ \ \{x[i] := a[i];\ \ \ i := i + 1;\ \} \\
:\ \mathit{comm}
\end{array}
$$

Fig. 2. Implementation of bubble-sort algorithm

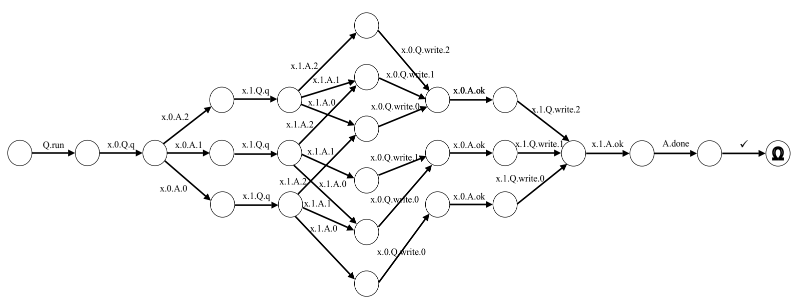

Fig. 3. A transition system of the model for n=2

Table 2. Experimental results for minimal model generation

n	CSP			Regular expressions		
	Time (min)	Max. states	Model states	Time (min)	Max. states	Model states
5	6	1 775	164	5	3 376	163
10	20	18 752	949	10	64 776	948
15	50	115 125	2 859	120	352 448	2 858
20	110	378 099	6 394	240	1 153 240	6 393
30	750	5 204 232	20 339	failed		

7 Conclusion

We presented a compositional representation of game semantics of an interesting fragment of Idealised Algol by CSP processes. This enables observational equivalence and a range of properties of terms-in-context (i.e. open program fragments) to be checked using the FDR tool.

We also reported initial experimental results using our prototype compiler and FDR. They show that, for minimal model generation, the CSP approach outperforms the approach based on regular expressions.

As future work, we plan to compare how the two approaches perform on a range of equivalence and property checking problems.

We also intend to extend the compiler so that parameterised IA terms (such as parametrically polymorphic programs) are translated to single parameterised CSP processes. Such processes could then be analysed by techniques which combine CSP and data specification formalisms (e.g. [5, 14]) or by algorithms based on data independence [11].

References

1. S.Abramsky and G.McCusker. Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In P.W.O'Hearn and R.D.Tennent, editors, *Algol-like languages*. Birkhaüser, 1997.
2. S.Abramsky. Algorithmic game semantics: A tutorial introduction. Lecture notes, Marktoberdorf International Summer School 2001, 2001.
3. S.Abramsky, D.Ghica, A.Murawski and C.-H.L.Ong. Applying Game Semantics to Compositional Software Modeling and Verifications. In Proceedings of TACAS, LNCS 2988, March 2004.
4. T.Ball and S.K.Rajamani, The SLAM Project: Debugging System Software via Static Analysis. In Proceedings of POPL, January 2002.
5. A.Farias, A.Mota and A.Sampaio. Efficient CSP-Z Data Abstraction. In Proceedings of IFM, LNCS 2999, April 2004.
6. Formal Systems (Europe) Ltd. Failures-Divergence Refinement: FDR2 Manual. 2000.
7. D.Ghica and G.McCusker. The Regular-Language Semantics of Second-order Idealized Algol. Theoretical Computer Science 309(1–3): 469–502, 2003.

8. D.Ghica. A Games-Based Foundation for Compositional Software Model Checking. PhD Thesis, Queen's University School of Computing, Kingston, Ontario, Canada, November 2002.
9. D.Ghica. Game-based Software Model Checking: Case Studies and Methodological Considerations. Oxford University Computing Lab, Technical Report PRG-RR-03-11, May 2003.
10. J.M.E.Hyland and C.-H.L.Ong. On full abstraction for PCF: I, II and III. Information and Computation 163: 285–408, 2000.
11. R.Lazić. A Semantic Study of Data Independence with Applications to Model Checking. DPhil thesis, Computing Laboratory, Oxford University, 1999.
12. C.-H.L.Ong. Observational equivalence of 3rd-order Idealised Algol is Decidable. In Proceedings of LICS, IEEE, July 2002.
13. J.C.Reynolds. The essence of Algol. In Proceedings of ISAL, 345–372, Amsterdam, Holland, 1981.
14. M.Roggenbach. CSP-CASL — A new Integration of Process Algebra and Algebraic Specification. In Proceedings of AMiLP, TWLT 21, Universiteit Twente, 2003.
15. A.W.Roscoe, P.H.B. Gardiner, M.H.Goldsmith, J.R.Hulance, D.M.Jackson and J.B.Scattergod. Hierarchical compression for model-checking CSP or how to check 10^{20} dining philosophers for deadlock. In Proceedings of TACAS, LNCS 1019, May 1995.
16. A.W.Roscoe. The Theory and Practice of Concurrency. Prentice Hall, 1998.