Verifying Data Independent Programs Using Game Semantics

Aleksandar S. Dimovski

Faculty of Information-Communication Tech., FON University, Macedonia

Abstract. We address the problem of verification of program terms parameterized by a data type X, such that the only operations involving X a program can perform are to input, output, and assign values of type X, as well as to test for equality such values. Such terms are said to be data independent with respect to X. Logical relations for game semantics of terms are defined, and it is shown that the Basic Lemma holds for them. This proves that terms are predicatively parametrically polymorphic, and it provides threshold collections, i.e. sufficiently large finite interpretations of X, for the problem of verification of observational-equivalence, approximation, and safety of parameterized terms for all interpretations of X. In this way we can verify terms with data independent infinite integer types. The practicality of the approach is evaluated on several examples.

1 Introduction

In this paper we study predicative parametric polymorphism in the setting of game semantics, and its applications to parameterized verification. In order to keep the presentation focussed, we work with Idealized Algol (IA) [1], an expressive programming language combining imperative features, locally-scoped variables and (call-by-name) higher-order functions.

Parametric polymorphism is the notion of treating data from a range of types in a uniform fashion. Its predicative version allows types to be formed from type variables, while its impredicative version is more general and allows universal quantification of type variables. We achieve predicative parametric polymorphism by extending our language with free data type variables X on which only the equality operation is available. Thus, any program term makes sense if any data type is instantiated for X, i.e. terms become parameterized by X. We will want to verify observational-equivalence, approximation, and safety of predicatively parametrically polymorphic terms.

We obtain results which provide threshold collections, i.e. sufficient finite interpretations of the data type variable X, such that if a property holds/fails for those interpretations, then it holds/fails for all interpretations which assign larger sets to the parameter X. Considering the case when an infinite integer type is substituted for X, we obtain a procedure to perform verification of terms which contain infinite integers completely automatically. This is done by replacing integers with threshold collections, i.e. appropriate small finite data types.

W. Binder, E. Bodden, and W. Löwe (Eds.): SC 2013, LNCS 8088, pp. 128–143, 2013.

A useful tool for enabling parameterized verification as above are logical relations [13,14,16]. A logical relation of a language is an assignment of relations to all types such that, the relation for any type is obtained from the relations assigned to data types and type variables, by induction on the structure of types. We will define logical relations for game semantics by lifting the relations on data values through all the constructs of game semantics, in such a way that the Basic Lemma holds for them. It states that logical relations are preserved by game semantics of terms. This proves that the language we consider is parametrically polymorphic, i.e. any term behaves uniformly for different instances of its parameter. The Basic Lemma will be applied to parameterized verification, since it provides direct ways of relating various game semantics interpretations of parameterized terms.

Game semantics [1,2] is a method for compositional modeling of programming languages, which constructs models of terms (open programs) by looking at the ways in which a term can observably interact with its environment. Types are interpreted by *games* (or arenas) between a Player, which represents the term being modelled, and an Opponent, which represents the environment in which the term is used, while terms are interpreted by *strategies* on games. Game semantics is compositional, i.e. defined recursively on the syntax, therefore the model of a larger term is constructed from the models of its constituting subterms, using a notion of strategy composition. Another important feature of this method, also known as external compositionality, is that there is a model for any term-in-context (open program) with undefined identifiers, such as calls to library functions. These two features are essential for achieving modular analysis of larger terms. The model obtained by game semantics is *fully abstract*, which means that it is both sound and complete with respect to observational equivalence of programs, and so it is the most accurate model we can use for a programming language. Although this model is precise, it is complicated and so equivalence and a range of properties are not decidable within it. However, it has been shown that for several language fragments with finite data types, the model can be given certain kinds of concrete automata-theoretic representations [4,5,6,12]. This gives a decision procedure for a range of verification problems, such as observational-equivalence, approximation, safety, and others, to be solved algorithmically.

The paper is organised as follows. Section 2 introduces the language considered in this paper, and its game semantics is defined in Section 3. Logical relations for game semantics are presented in Section 4. Several theorems which provide support for parameterized verification are shown in Section 5. The practicality of this approach is demonstrated in Section 6. In Section 7, we conclude and discuss possible extensions.

Related Work. Predicative parametric polymorphism is known as data independence in the setting of concurrent reactive systems. Some practically important examples of such systems are communication protocols, memory systems, and security protocols. The literature contains efficient algorithms for deciding the parameterized verification problem for data independent systems [10,11].

System F is a typical example of impredicative parametric polymorphism. It is also known as the polymorphic or second-order λ-calculus, and it works with pure type theories, with no notion of ground data types. Game semantics for System F are given in [8,9].

2 Programming Language

Idealized Algol (IA) [1,2] is a simply-typed call-by-name λ-calculus with the fundamental imperative features and locally-scoped variables. We extend IA with predicative parametric polymorphism by allowing data type variables X.

The data types D are finite integers ($\mathsf{int}_n = \{0, \ldots, n-1\}$), booleans, and a data type variable X ($D ::= \mathsf{int}_n \mid \mathsf{bool} \mid X$). The phrase types consists of base types: expressions, commands, variables ($B ::= \mathsf{exp}D \mid \mathsf{var}D \mid \mathsf{com}$) and function types ($T ::= B \mid T \to T$).

Terms of the language are the standard functional constructs for function definition ($\lambda\, x : T.M$) and application (MN) as well as recursion (YM). Expression constants are integers (n), booleans (tt, ff), and data values from the set W which interprets X ($w \in W$). The usual arithmetic-logic operations are employed ($M \,\mathsf{op}\, N$), but equality is the only operation available on X-expressions. We have the usual imperative constructs: sequential composition ($M; N$), conditional ($\mathsf{if}\, M\, \mathsf{then}\, N\, \mathsf{else}\, N'$), iteration ($\mathsf{while}\, M \,\mathsf{do}\, N$), assignment ($M := N$), de-referencing ($!M$), "do nothing" command skip, and $\mathsf{diverge}$ which represents an infinite loop (divergence). Block-allocated local variables are introduced by a *new* construct ($\mathsf{new}_D\, x := v\, \mathsf{in}\, M$), which initializes a variable and makes it local to a given block. The constructor $\mathsf{mkvar}_D MN$ is used for creating "bad" variables.

Well-typed terms are given by typing judgements of the form $\Gamma \vdash_W M : T$, where Γ is a type *context* consisting of a finite number of typed free identifiers, and W is a set of data values used to interpret X, which are allowed to occur in M as expression constants. When it does not cause ambiguity, we may write only $\Gamma \vdash M : T$. Typing rules of the language are those of IA (e.g. [1,2]), where the rules for arithmetic-logic operations are:

$$\frac{\Gamma \vdash_W M : \mathsf{exp}D \qquad \Gamma \vdash_W N : \mathsf{exp}D}{\Gamma \vdash_W M \,\mathsf{op}\, N : \mathsf{exp}D'} \qquad \frac{\Gamma \vdash_W M : \mathsf{exp}X \qquad \Gamma \vdash_W N : \mathsf{exp}X}{\Gamma \vdash_W M = N : \mathsf{expbool}}$$

where $D, D' \in \{\mathsf{int_n}, \mathsf{bool}\}$, and $\mathsf{op} \in \{+, -, *, /, =, \neq, <, >, \wedge, \vee, \neg\}$. For such terms we say that are *data independent* with respect to the data type X.

Any well-typed term can contain equality tests between values of X. We define a condition on terms, which does not allow any equality tests between values of X. A term $\Gamma \vdash_W M : T$ satisfies $\mathbf{(NoEq}_X\mathbf{)}$ condition if for any equality operation $\Gamma' \vdash_W N = N'$ within M, X does not occur in the types of N and N', i.e. $\Gamma' \vdash_W N, N' : \mathsf{exp}\{\mathsf{int_n}, \mathsf{bool}\}$.

The operational semantics of our language is given for terms $\Gamma \vdash_W M : T$, such that all identifiers in Γ are variables, i.e. $\Gamma = x_1 : \mathsf{var}D_1, \ldots, x_k : \mathsf{var}D_k$. It is defined by a big-step reduction relation: $\Gamma \vdash_W M, \mathrm{s} \Longrightarrow V, \mathrm{s}'$, where s, s$'$

represent Γ-*states* before and after reduction. A Γ-state s is a (partial) function assigning data values to the variables $\{x_1, \ldots, x_k\}$. We denote by V terms in *canonical form* defined by $V ::= x \mid v \mid \lambda\, x : T.M \mid \mathsf{skip} \mid \mathsf{mkvar}_D MN$. Reduction rules are those of IA [1,2].

Given a term $\Gamma \vdash_W M : \mathsf{com}$, where all identifiers in Γ are variables, we say that M *terminates* in state s, if $\Gamma \vdash_W M, \mathrm{s} \Longrightarrow \mathsf{skip}, \mathrm{s}'$ for some state s'. Then, we say that a term $\Gamma \vdash_W M : T$ is an *approximation* of a term $\Gamma \vdash_W N : T$, denoted by $\Gamma \vdash_W M \sqsubseteq N$, if and only if for any term-with-hole [1] $C[-] : \mathsf{com}$, such that both $C[M]$ and $C[N]$ are well-typed terms of type com, if $C[M]$ terminates then $C[N]$ terminates. If two terms approximate each other they are considered *observationally-equivalent*, denoted by $\Gamma \vdash_W M \cong N$.

3 Game Semantics

We now give a brief description of game semantics for IA extended with predicative parametric polymorphism. A more detailed presentation of game semantics for IA can be found in [1,2].

An *arena* A is a triple $\langle M_A, \lambda_A, \vdash_A \rangle$, where M_A is a countable set of *moves*, $\lambda_A : M_A \to \{\mathsf{O}, \mathsf{P}\} \times \{\mathsf{Q}, \mathsf{A}\}$ is a labeling function which indicates whether a move is by *Opponent* (O) or *Player* (P), and whether it is a *question* (Q) or an *answer* (A). Then, $\vdash_A$ is a binary relation between $M_A + \{*\}$ $(* \notin M_A)$ and M_A, called *enabling* (if $m \vdash_A n$ we say that m enables move n), which satisfies the following conditions: (i) Initial moves (a move enabled by $*$ is called *initial*) are Opponent questions, and they are not enabled by any other moves besides $*$; (ii) Answer moves can only be enabled by question moves; (iii) Two participants always enable each others moves, never their own.

We denote the set of all initial moves in A as I_A. The simplest arena is the empty arena $I = \langle \emptyset, \emptyset, \emptyset \rangle$. Given arenas A and B, we define new arenas $A \times B$, $A \Rightarrow B$ as follows:

$$A \times B = \langle\, M_A + M_B,\, [\lambda_A, \lambda_B],\, \vdash_A + \vdash_B \,\rangle$$
$$A \Rightarrow B = \langle\, M_A + M_B,\, [\overline{\lambda}_A, \lambda_B],\, \vdash_B + \big(I_B \times I_A\big) + \big(\vdash_A \cap (M_A \times M_A)\big)\rangle$$

where $+$ is a disjoint union, and $\overline{\lambda}_A$ is like λ_A except that it reverses O/P part of moves while preserving their Q/A part.

Let W be an arbitrary set of data values, and w be a meta-variable ranging over W. A *parameterized arena* $A_W = \langle M_{A_W}, \lambda_{A_W}, \vdash_{A_W} \rangle$ is defined as follows. The set of moves M_{A_W} is of the form $C_A \cup (P_A \times W)$, where C_A is a set of constant moves that do not depend on W, and P_A is a set of parameterized move-tags so that for any $p \in P_A$ and $w \in W$, (p, w) is a move. Moves of the form (p, w) are called parameterized moves, and we will also denote them as $p(w)$. Each particular parameterized move-tag $p \in P_A$ will generate one partition of M_{A_W},

[1] A term-with-hole $C[-] : \mathsf{com}$ is a term with with zero or more holes $[-]$ in it, such that if $\Gamma \vdash M : T$ is a term of the same type as the hole then $C[M]$ is a well-typed closed term of type com, i.e. $\vdash C[M] : \mathsf{com}$.

denoted as $[p] = \{p(w) \mid w \in W\}$. The partition of M_{A_W} induced by a constant move $c \in C_A$ is a singleton set $[c] = \{c\}$. The partitioning of M_{A_W} induced by W is: $\{[c] \mid c \in C_A\} \cup \{[p] \mid p \in P_A\}$.

All moves of M_{A_W} that belong to a single partition of the form $[p]$ have the same labellings and enablings, i.e. $\lambda_{A_W}(p(w))$ is the same for all $w \in W$, and if $(p(w), n) \in \vdash_{A_W}$ (resp., $(n, p(w)) \in \vdash_{A_W}$) for some $n \in M_{A_W}, w \in W$, then $[p] \times \{n\} \subseteq \vdash_{A_W}$ (resp., $\{n\} \times [p] \subseteq \vdash_{A_W}$).

Now we are ready to give interpretations of the types of our language. The data types are interpreted by sets of values they can contain:

$$[\![int_n]\!]_W = \{0, \ldots n-1\} \qquad [\![bool]\!]_W = \{tt, ff\} \qquad [\![X]\!]_W = W$$

The base types are interpreted by parameterized arenas, where all questions are initial and P-moves answer them.

$$\begin{array}{l}
[\![\mathsf{expD}]\!]_W = \langle\, \{q, v \mid v \in [\![\mathsf{D}]\!]_W\}, \{\lambda(q) = \mathsf{OQ},\ \lambda(v) = \mathsf{PA}\}, \\
\qquad\qquad \{(*, q),\ (q, v) \mid v \in [\![\mathsf{D}]\!]_W\} \,\rangle \\
[\![\mathsf{com}]\!]_W = \langle \{run, done\}, \{\lambda(run) = \mathsf{OQ}, \lambda(done) = \mathsf{PA}\}, \{(*, run), (run, done)\} \rangle \\
[\![\mathsf{varD}]\!]_W = \langle\, \{read, v, write(v), ok \mid v \in [\![\mathsf{D}]\!]_W\}, \{\lambda(read, write(v)) = \mathsf{OQ}, \\
\lambda(v, ok) = \mathsf{PA}\}, \{(*, read), (*, write(v)), (read, v), (write(v), ok) \mid v \in [\![\mathsf{D}]\!]_W\} \rangle
\end{array}$$

In the arena for expressions, there is an initial move q to ask for the value of the expression, and corresponding to it a value from $[\![D]\!]_W$. Note that, the set of moves of $[\![\mathsf{expX}]\!]_W$ has two partitions $\{q\}$ and $\{v \mid v \in W\}$. For commands, there is an initial move *run* to initiate a command, and an answer move *done* to signal successful termination of a command. This arena does not depend on W. In the arena for variables, we have moves for writing to the variable, *write*(v), acknowledged by the move *ok*, and for reading from the variable, a move *read*, and corresponding to it a value from $[\![D]\!]_W$. $M_{[\![\mathsf{varX}]\!]_W}$ has four partitions $\{read\}$, $\{ok\}$, $\{v \mid v \in W\}$, and $\{write(v) \mid v \in W\}$.

A *justified sequence* s_W in arena A_W is a finite sequence of moves of A_W together with a pointer from each non-initial move n to an earlier move m such that $m \vdash_{A_W} n$. We say that n is (explicitly) justified by m, or when n is an answer that n *answers* m. A *legal play* (or play) is a justified sequence with some additional constraints: *alternation* (Opponent and Player moves strictly alternate), *well-bracketed* condition (when an answer is given, it is always to the most recent question which has not been answered), and *visibility* condition (a move to be played is justified by a move from a certain subsequence of the play so far, called view). The set of all legal plays in arena A_W is denoted by L_{A_W}.

A *strategy* σ_W on an arena A_W (written as $\sigma_W : A_W$) is a non-empty set of even-length plays of A_W satisfying: if $s_W \cdot m \cdot n \in \sigma_W$ then $s_W \in \sigma_W$; and if $s_W \cdot m \cdot n, s_W \cdot m \cdot n' \in \sigma_W$ then $n = n'$. A strategy specifies what options Player has at any given point of a play and it does not restrict the Opponent moves. A play is *complete* if all questions occurring in it have been answered. Given a strategy σ_W, we define the corresponding *complete strategy* σ_W^{comp} as the set of its non-empty complete plays. We write $Str_{A_W}^{comp}$ for the set of all complete strategies for the arena A_W.

Composition of strategies is interpreted as CSP-style "parallel composition plus hiding". Given strategies $\sigma_W : A_W \Rightarrow B_W$ and $\tau_W : B_W \Rightarrow C_W$, the *composition* $\sigma_W \fatsemi \tau_W : A_W \Rightarrow C_W$ consists of sequences generated by playing σ_W and τ_W in parallel, making them synchronize on moves in the shared arena B_W. Moves in B_W are subsequently hidden. We now formally define composition of strategies. Let u be a sequence of moves from A_W, B_W, and C_W. We define $u \upharpoonright B_W, C_W$ to be the subsequence of u obtained by deleting all moves from A_W along with all associated pointers from/to moves of A_W. Similarly define $u \upharpoonright A_W, B_W$. Define $u \upharpoonright A_W, C_W$ to be the subsequence of u consisting of all moves from A_W and C_W, but where there was a pointer from a move $m_A \in M_{A_W}$ to an initial move $m \in I_{B_W}$ extend the pointer to the initial move in C_W which was pointed to from m. We say that u is an *interaction* of A_W, B_W, C_W if $u \upharpoonright A_W, B_W \in L_{A_W \Rightarrow B_W}$, $u \upharpoonright B_W, C_W \in L_{B_W \Rightarrow C_W}$, and $u \upharpoonright A_W, C_W \in L_{A_W \Rightarrow C_W}$. The set of all such sequences is written as $int(A_W, B_W, C_W)$. We define:

$$\sigma_W \fatsemi \tau_W = \{u \upharpoonright A_W, C_W \mid u \in int(A_W, B_W, C_W) \wedge u \upharpoonright A_W, B_W \in \sigma_W \wedge u \upharpoonright B_W, C_W \in \tau_W\}$$

The *identity strategy* $id_{A_W} : A_W \Rightarrow A_W$, which is also called *copy-cat*, is defined in such a way that a move by Opponent in either occurrence of A_W is immediately copied by Player to the other occurrence, i.e. we have

$$id_{A_W} = \{s \in L_{A^l_W \Rightarrow A^r_W} \mid \forall\, s' \sqsubseteq^{even} s.\ s' \upharpoonright A^l_W = s' \upharpoonright A^r_W\}$$

where the l and r tags are used to distinguish between the two occurrences of A, $s' \sqsubseteq^{even} s$ means that s' is an even-length prefix of s, and $s' \upharpoonright A^l_W$ is the subsequence of s' consisting of all moves from A^l_W.

Plays in a strategy may contain several occurrences of initial moves, which define different *threads* inside plays in the following way: a thread is a subsequence of a play whose moves are connected via chains of pointers to the same occurrence of an initial move. We consider the class of *single-threaded* strategies whose behaviour depends only on one thread at a time, i.e. any Player move depends solely on the current thread of the play. We say that a strategy is *well-opened* if all its plays have exactly one initial move. It can be established one-to-one correspondence between single-threaded and well-opened strategies. The set of all strategies for an arena forms a complete partial order (cpo) under the inclusion order ($\subseteq$). The least element is $\{\epsilon\}$, and the least upper bound is given by unions. It is shown in [1,2] that arenas as objects and single-threaded (well-opened) strategies as arrows constitute a cpo-enriched cartesian closed category. From now on, we proceed to work only with well-opened strategies.

A type T is interpreted as an arena $[\![T]\!]_W$, and a term $\Gamma \vdash_W M : T$, where $\Gamma = x_1 : T_1, \ldots, x_n : T_n$, is interpreted by a strategy $[\![\Gamma \vdash M : T]\!]_W$ for the arena $[\![\Gamma \vdash T]\!]_W = [\![T_1]\!]_W \times \ldots \times [\![T_n]\!]_W \Rightarrow [\![T]\!]_W$. Language constants and constructs are interpreted by strategies and compound terms are modelled by composition of the strategies that interpret their constituents. Identity strategies are used to interpret free identifiers from Γ. Some of the strategies [1,2] are given

below, where to simplify representation of plays, every move is tagged with the index of type component where it occurs.

$[\![\mathsf{v} : \mathsf{exp}D]\!]_W^{comp} = \{q\, v\}$ $[\![\mathsf{skip} : \mathsf{com}]\!]_W^{comp} = \{run\, done\}$ $[\![\mathsf{diverge} : \mathsf{com}]\!]_W^{comp} = \emptyset$
$[\![\mathsf{op} : \mathsf{exp}D^1 \times \mathsf{exp}D^2 \to \mathsf{exp}D]\!]_W^{comp} = \{q\, q^1\, v^1\, q^2\, v'^2\, (v\ op\ v') \mid v, v' \in [\![D]\!]_W\}$
$[\![; : \mathsf{com}^1 \times \mathsf{com}^2 \to \mathsf{com}]\!]_W^{comp} = \{run\, run^1\ done^1\ run^2\ done^2\ done\}$
$[\![:=: \mathsf{var}D^1 \times \mathsf{exp}D^2 \to \mathsf{com}]\!]_W^{comp} = \{run\, q^2\, v^2\, write(v)^1\, ok^1\, done \mid v \in [\![D]\!]_W\}$

Using standard game-semantic techniques, it can be shown as in [1,2] that this model is fully abstract for observational-equivalence.

Theorem 1. *$\Gamma \vdash_W M \subseteq_\approx N$ iff $[\![\Gamma \vdash M]\!]_W^{comp} \subseteq [\![\Gamma \vdash N]\!]_W^{comp}$.*

Suppose that there is a special free identifier abort of type com in Γ. Let $M[N/x]$ denote the capture-free substitution of N for x in M. We say that a term $\Gamma \vdash_W M$ is *safe* iff $\Gamma \backslash \mathsf{abort} \vdash_W M[\mathsf{skip}/\mathsf{abort}] \subseteq_\approx M[\mathsf{diverge}/\mathsf{abort}]$; otherwise we say that a term is *unsafe*. Since the game-semantics model is fully abstract, the following can be shown (see also [3]).

Lemma 1. *A term $\Gamma \vdash_W M$ is safe if $[\![\Gamma \vdash M]\!]_W^{comp}$ does not contain any play with moves from $M_{[\![\mathsf{com}^{abort}]\!]}$.*

For example, $[\![\mathsf{abort} : \mathsf{com}^{\mathsf{abort}} \vdash \mathsf{skip}\,;\ \mathsf{abort} : \mathsf{com}]\!]^{comp}$ is the set $\{run \cdot run^{abort} \cdot done^{abort} \cdot done\}$, so this term is unsafe.

4 Logical Relations

A binary *relation* between sets W_0 and W_1 is any subset $R \subseteq W_0 \times W_1$. We will use the notation $R : W_0 \longleftrightarrow W_1$ to mean that R is a binary relation between W_0 and W_1, and $w_0\, R\, w_1$ to mean $(w_0, w_1) \in R$, in which case we say that w_0 and w_1 are R-related. The domain of a relation R is the set of the first components of all pairs in R. We say that R is a *partial function* iff $\forall\, w_0, w_1, w_1'.(w_0 R w_1 \wedge w_0 R w_1') \Rightarrow w_1 = w_1'$, and R is *injective* iff $\forall\, w_0, w_0', w_1.(w_0 R w_1 \wedge w_0' R w_1) \Rightarrow w_0 = w_0'$. A special case of relation is the *identity relation* $I_W : W \longleftrightarrow W$, defined by $I_W = \{(w, w) \mid w \in W\}$, i.e. $w\, I_W\, w'$ iff $w = w'$. Next, we define relations on sequences. For any $R : W_0 \longleftrightarrow W_1$, define $R^* : W_0^* \longleftrightarrow W_1^*$ as

$$t\, R^*\, t' \quad \text{iff} \quad t_1\, R\, t_1' \wedge \ldots \wedge t_{|t|}\, R\, t'_{|t'|}$$

where $|\ t\ |$ denotes the length of t, and for any $1 \leq k \leq |\ t\ |$, t_k denotes the k-th element of t. That is, sequences are R-related if they have the same length and corresponding elements are R-related.

Let $R : W_0 \longleftrightarrow W_1$ be a relation. For any data type, we define the relation $[\![D]\!]_{R, W_0, W_1} : [\![D]\!]_{W_0} \longleftrightarrow [\![D]\!]_{W_1}$ as follows.

$$[\![int_n]\!]_{R, W_0, W_1} = I_{[\![int_n]\!]} \quad [\![bool]\!]_{R, W_0, W_1} = I_{[\![bool]\!]} \quad [\![X]\!]_{R, W_0, W_1} = R$$

Next we "lift" the definition of relations to arenas. We define a relational arena $A_{R, W_0, W_1} = \langle M_{A_{R, W_0, W_1}}, \lambda_{A_{R, W_0, W_1}}, \vdash_{A_{R, W_0, W_1}} \rangle : A_{W_0} \longleftrightarrow A_{W_1}$ between two

parameterized arenas induced by R. Let $M_{A_{W_i}} = C_A \cup (P_A \times W_i)$ for $i = 0, 1$. Then, we have:

$$\begin{array}{ll} (m, m') \in M_{A_{R,W_0,W_1}} \text{ iff} & \begin{cases} m = m', \text{ if } m \in C_A \\ w\, R\, w', \text{ if } m \in P_A \times W_0, m = p(w), m' = p(w') \end{cases} \\ \lambda_{A_{R,W_0,W_1}}(m, m') = & \lambda_{A_{W_0}}(m) = \lambda_{A_{W_1}}(m') \\ * \vdash_{A_{R,W_0,W_1}} (m, m') \text{ iff} & (* \vdash_{A_{W_0}} m \wedge * \vdash_{A_{W_1}} m') \\ (n, n') \vdash_{A_{R,W_0,W_1}} (m, m') & \text{iff } (n \vdash_{A_{W_0}} m \wedge n' \vdash_{A_{W_1}} m') \end{array}$$

Let define a relation $L_{A_{R,W_0,W_1}} : L_{A_{W_0}} \longleftrightarrow L_{A_{W_1}}$ between the sets of all legal plays in A_{W_0} and A_{W_1} induced by R.

$$\begin{array}{l} s\, L_{A_{R,W_0,W_1}}\, s' \text{ iff } (i) s\, (M_{A_{R,W_0,W_1}})^*\, s' \\ \qquad\qquad (ii)\, \lambda_{A_{W_0}}(s_i) = \lambda_{A_{W_1}}(s'_i), \text{ for } 1 \leq i \leq |\, s\, | \\ \qquad\qquad (iii) s_i \text{ justifies } s_j \text{ iff } s'_i \text{ justifies } s'_j, \text{ for } 1 \leq i < j \leq |\, s\, | \end{array} \quad (1)$$

We define $Domain(L_{A_{R,W_0,W_1}})$ as the set of all legal plays s from A_{W_0}, such that for any parameterized move $p(w)$ in s we have that w is in the domain of R.

Finally, we define a relation $Str^{comp}_{A_{R,W_0,W_1}} : Str^{comp}_{A_{W_0}} \longleftrightarrow Str^{comp}_{A_{W_1}}$ between complete strategies on A_{W_0} and A_{W_1} induced by R.

$$\begin{array}{l} \sigma\, Str^{comp}_{A_{R,W_0,W_1}}\, \sigma' \text{ iff } \forall\, s \in \sigma. s \in Domain(L_{A_{R,W_0,W_1}}) \Rightarrow \exists\, S' \subseteq \sigma'. S' \neq \emptyset \wedge \\ \qquad\qquad (\forall\, s' \in S'. s\, L_{A_{R,W_0,W_1}}\, s') \wedge Closed(A, R, W_0, W_1, s, S') \end{array} \quad (2)$$

$$\begin{array}{l} Closed(A, R, W_0, W_1, s, S') = \forall\, s' \in S'. \forall\, k. \forall\, w' \in W_1. \lambda^{OP}(s_k) = O \wedge \\ \quad s_k = p(w) \wedge w\, R\, w' \Rightarrow \exists\, s'' \in S'. s''_1 = s'_1 \wedge \ldots \wedge s''_{k-1} = s'_{k-1} \wedge s''_k = p(w') \end{array}$$

That is, two complete strategies σ and σ' are R-related if and only if for any complete play s from σ which is in the domain of the logical relation, there exists a nonempty subset S' of σ' such that s is R-related to any complete play in S' and S' is closed under those choices of Opponent moves which preserve the relation R. We say that S' is *R-closed* with respect to s.

Before we prove the Basic Lemma for logical relations, we first show several useful technical lemmas.

Lemma 2. *Let $\sigma_W : A_W \Rightarrow B_W$ and $\tau_W : B_W \Rightarrow C_W$ be two strategies and $R : W_0 \longleftrightarrow W_1$ be a relation. If we have that $\sigma^{comp}_{W_0}\, (Str^{comp}_{A \Rightarrow B_{R,W_0,W_1}})\, \sigma^{comp}_{W_1}$ and $\tau^{comp}_{W_0}\, (Str^{comp}_{B \Rightarrow C_{R,W_0,W_1}})\, \tau^{comp}_{W_1}$, then*

$$(\sigma_{W_0} \fatsemi \tau_{W_0})^{comp}\ (Str^{comp}_{A \Rightarrow C_{R,W_0,W_1}})\ (\sigma_{W_1} \fatsemi \tau_{W_1})^{comp}$$

Proof. Let $s \in (\sigma_{W_0} \fatsemi \tau_{W_0})^{comp}$. Then there must be some witness to this, i.e. some $u \in int(A_{W_0}, B_{W_0}, C_{W_0})$ such that $s = u \upharpoonright A_{W_0}, C_{W_0}$. By definition of composition, we have that $u \upharpoonright B_{W_0}, C_{W_0} \in \tau^{comp}_{W_0}$, and for every B_W-initial move $i \in u \upharpoonright B_{W_0}, C_{W_0}$ we have $(u \upharpoonright A_{W_0}, B_{W_0}) \upharpoonright i \in \sigma^{comp}_{W_0}$. We denote by $s \upharpoonright i$ the subsequence of s consisting of all those moves which are hereditarily justified (via chains of pointers) by the same initial move i. Since σ_{W_0} is R-related to σ_{W_1}, there is a set $U_i \subseteq \sigma^{comp}_{W_1}$ such that U_i is R-closed with respect

to $(u \restriction A_{W_0}, B_{W_0}) \restriction i$ for any B_W-initial move i in $u \restriction A_{W_0}, B_{W_0}$. Since τ_{W_0} is R-related to τ_{W_1}, there is a set $U \subseteq \tau_{W_1}^{comp}$ such that U is R-closed with respect to $u \restriction B_{W_0}, C_{W_0}$. We can now generate a set U'. For an arbitrary $u_1 \in U$ we create a set of sequences in U' as follows. For any B_W-initial move $i \in u_1$, we choose an arbitrary $u_{i,1} \in U_i$ such that $(u_1 \restriction B) \restriction i = u_{i,1} \restriction B$. Then we generate an interaction sequence $u' \in int(A_{W_1}, B_{W_1}, C_{W_1})$, such that $u' \restriction B_{W_1}, C_{W_1} = u_1$, and $(u' \restriction A_{W_1}, B_{W_1}) \restriction i = u_{i,1}$. We repeat this process for all possible $u_1 \in U$ and $u_{i,1} \in U_i$ for any B_W-initial move $i \in u_1$. Finally, we obtain $S = \{u' \restriction A_{W_1}, C_{W_1} \mid u' \in U'\}$, which is R-closed with respect to s. □

Lemma 3. *Let $id_{A_W} : A_W \Rightarrow A_W$ be an identity strategy and $R : W_0 \longleftrightarrow W_1$ be a relation. Then $id_{A_{W_0}}^{comp}\,(Str_{A \Rightarrow A_{R,W_0,W_1}}^{comp})\,id_{A_{W_1}}^{comp}$.*

Proof. Let $s \in id_{A_{W_0}}^{comp}$. We first generate a complete play $t \in id_{A_{W_1}}^{comp}$, such that s is R-related to t. This is done by choosing for any Opponent move s_k of the form $p(w)$, where k is odd, a R-related move $p(w')$ from $M_{A_{W_1}}$, and setting $t_k = p(w')$, $t_{k+1} = p(w')$. Then we obtain a set S' from t, such that for any odd k and for any $m \in M_{A_{W_1}}$, where $s_k\, M_{R,A_{W_0},A_{W_1}}\, m$, we create a sequence in S': $s' = t_1 \dots t_{k-1} m m t_{k+2} \dots t_{|t|}$. Such S' is R-closed with respect to s. □

Let R be a relation between two cpo $(V, \leq)$ and $(V', \leq')$, and let $\precsim$ be the pointwise ordering on R. We say that R is *complete* iff $(R, \precsim)$ is cpo such that:

- the least element is $(\bot, \bot')$, where $\bot$ (resp., $\bot'$) is the least element of $(V, \leq)$ (resp., $(V', \leq')$).
- for any directed set $D \subseteq R$, its least upper bound consists of pointwise least upper bounds of $(V, \leq)$ and $(V', \leq')$.

Lemma 4. *For any parameterized arena A_W and for any relation $R : W_0 \longleftrightarrow W_1$, we have that $Str_{A_{R,W_0,W_1}}^{comp}$ is complete.*

Proof. The least elements of $Str_{A_{W_0}}^{comp}$ and $Str_{A_{W_1}}^{comp}$ are both $\emptyset$, and the least upper bounds are unions. □

We now present the Basic Lemma of logical relations for our language.

Theorem 2. *Let $\Gamma, z_1 : \mathsf{exp}X, \dots, z_k : \mathsf{exp}X \vdash M : T$ be a term such that M contains no data values of X, and $R : W_0 \longleftrightarrow W_1$ be a relation such that $(w_1, w_1'), \dots, (w_k, w_k')$ are some pairs in R. If either M satisfies ($\mathbf{NoEq}_X$) or R is a partial function and injective, then*

$$[\![\Gamma \vdash M[w_1/z_1, \dots, w_k/z_k]]\!]_{W_0}^{comp}(Str_{[\![\Gamma \vdash T]\!]_{R,W_0,W_1}}^{comp})[\![\Gamma \vdash M[w_1'/z_1, \dots, w_k'/z_k]]\!]_{W_1}^{comp}$$

Proof. The proof is by induction on the structure of terms.

The case of free identifiers holds due to the fact that logical relations are preserved by identity strategies (see Lemma 3).

Let $\Delta = z_1 : \mathsf{exp}X, \ldots, z_k : \mathsf{exp}X$. Consider the case $\Gamma, \Delta \vdash M = N$, where $\Gamma, \Delta \vdash M, N : \mathsf{exp}X$. By induction hypothesis, we have that:

$$[\![\Gamma \vdash M[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}\,(Str^{comp}_{[\![\Gamma \vdash \mathsf{exp}X]\!]_{R,W_0,W_1}})\,[\![\Gamma \vdash M[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1} \quad (*)$$
$$[\![\Gamma \vdash N[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}\,(Str^{comp}_{[\![\Gamma \vdash \mathsf{exp}X]\!]_{R,W_0,W_1}})\,[\![\Gamma \vdash N[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1} \quad (**)$$

where $w_1 R w'_1, \ldots, w_k R w'_k$, and we also use the following abbreviations: $\overrightarrow{w} = (w_1, \ldots, w_k)$, $\overrightarrow{w'} = (w'_1, \ldots, w'_k)$, and $M[\overrightarrow{w}/\Delta] = M[w_1/z_1, \ldots, w_k/z_k]$. Suppose $t \in [\![\Gamma \vdash M = N[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}$. Then either $t = q\,t_1\,t_2\,tt$ or $t = q\,t_1\,t_2\,ff$, where $q\,t_1\,w_1 \in [\![\Gamma \vdash M[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}$ and $q\,t_2\,w_2 \in [\![\Gamma \vdash N[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}$. Let $w_1 \neq w_2$, and so $t = q\,t_1\,t_2\,ff$. Since R is a partial function and injective, for any play $q\,t'_1\,w'_1 \in [\![\Gamma \vdash M[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1}$ related by R with $q\,t_1\,w_1$, and for any $q\,t'_2\,w'_2 \in [\![\Gamma \vdash N[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1}$ related by R with $q\,t_2\,w_2$, it must be that $w'_1 \neq w'_2$. Then all plays of the form $t' = q\,t'_1\,t'_2\,ff \in [\![\Gamma \vdash M = N[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1}$ are R-related with t. The other case, when $w_1 = w_2$ is similar. So we have $[\![\Gamma \vdash M = N[\overrightarrow{w}/\Delta]]\!]^{comp}_{W_0}\,(Str^{comp}_{[\![\Gamma \vdash \mathsf{expbool}]\!]_{R,W_0,W_1}})\,[\![\Gamma \vdash M = N[\overrightarrow{w'}/\Delta]]\!]^{comp}_{W_1}$.

Recursion $\Gamma \vdash YM : T$ is handled by using the following facts:

- $Str^{comp}_{[\![\Gamma \vdash T]\!]_{R,W_0,W_1}}$ is complete by Lemma 4
- the inductive hypothesis $[\![\Gamma \vdash M : T \to T]\!]^{comp}_{W_0}\,(Str^{comp}_{[\![\Gamma \vdash T \to T]\!]_{R,W_0,W_1}})\,[\![\Gamma \vdash M : T \to T]\!]^{comp}_{W_1}$
- and the definition of recursion $[\![\Gamma \vdash YM : T]\!]$ (see [1,2]).

The other cases for language constants and constructs are proved similarly, and we also use the fact shown in Lemma 2 that logical relations are preserved by composition of strategies. □

Remark. In Theorem 2, we have made an assumption that data values of X that occur as expression constants in M must be R-related in the two versions of M with parameters W_0 and W_1. This is so because any complete play s and any play in its R-closed set S' have to be R-related on their Player moves. For example, let $W_0 = W_1 = \{0, 1\}$, and $R = \{(0, 1)\}$. Then $[\![\vdash 0 : \mathsf{exp}X]\!]_{W_0}$ is not R-related to $[\![\vdash 0]\!]_{W_1}$, but is R-related to $[\![\vdash 1]\!]_{W_1}$.

Example 1. Consider the term M_1:

$$f : \mathsf{exp}X^{f,1} \to \mathsf{com}^f, x : \mathsf{exp}X^x \vdash f(x) : \mathsf{com}$$

The model representing this term, when $[\![X]\!] = \{0, 1\}$, is shown in Fig. 1. The model illustrates only the possible behaviors of this term: f may evaluate its argument, zero or more times, then the term terminates with a move *done*. The model makes no assumption about the number of times that f uses its argument. Note that moves tagged with f represent the actions of calling and returning from the function, while moves tagged with $f, 1$ are the actions caused by evaluating the first argument of f.

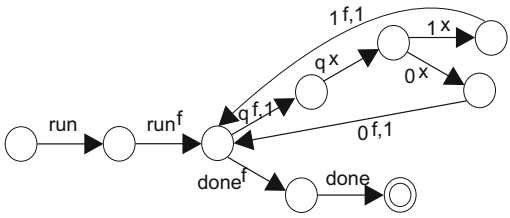

Fig. 1. The strategy for M_1 as a finite automaton

Let $W = \{0,1\}$, $W' = \mathbb{Z}$ (integers), and $R = \{(0,-n),(1,n) \mid n \in \{0,1,\ldots\}\}$. This term satisfies ($\mathbf{NoEq}_X$), so by Theorem 2 we have that $[\![M_1]\!]^{comp}_W$ and $[\![M_1]\!]^{comp}_{W'}$ are related by R. For example, let $s = run\, run^f\, q^{f,1}\, q^x\, 1^x\, 1^{f,1}\, done^f\, done \in [\![M_1]\!]^{comp}_W$. Then, the corresponding R-closed subset of $[\![M_1]\!]^{comp}_{W'}$ is:

$$S' = \{run\, run^f\, q^{f,1}\, q^x\, n^x\, n^{f,1}\, done^f\, done \mid n \in \{0,1,\ldots\}\}$$

Also note that, for $s = run\, run^f\, q^{f,1}\, q^x\, 1^x\, 1^{f,1}\, q^{f,1}\, q^x\, 1^x\, 1^{f,1}\, done^f\, done$, the corresponding R-closed set is:

$$S' = \{run\, run^f\, q^{f,1}\, q^x\, n^x\, n^{f,1}\, q^{f,1}\, q^x\, m^x\, m^{f,1}\, done^f\, done \mid\ n, m \in \{0,1,\ldots\}\}$$

This is so because two occurrences of 1^x in s are Opponent moves, so S' needs to be closed under all alternative choices of these moves which preserve R.

Let $W = \{0,1,2,3\}$, $W' = \{0\}$, and $R = \{(0,0)\}$. Then $[\![M_1]\!]^{comp}_W$ and $[\![M_1]\!]^{comp}_{W'}$ are also related by R. □

Example 2. Consider the term M_2:

$$f : \mathsf{com}^{f,1} \to \mathsf{com}^f, abort : \mathsf{com}^{abort}, x : \mathsf{exp}X^x, y : \mathsf{exp}X^y \vdash f\big(\mathsf{if}\ \neg(x = y)\ \mathsf{then}\ abort\big) : \mathsf{com}$$

The model for this term with parameter $\{tt, ff\}$ is shown in Fig. 2. Let $W = \{0,1,2,3\}$, $W' = \{tt, ff\}$, and $R = \{(0,tt),(1,ff)\}$. This term does not satisfy

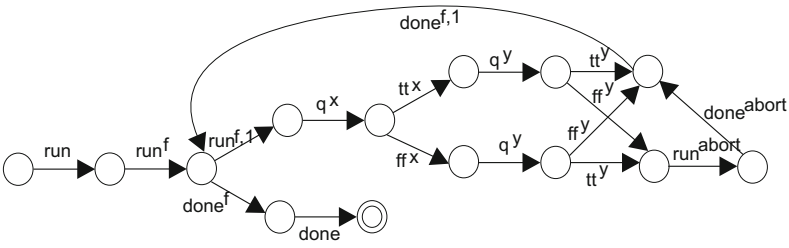

Fig. 2. The strategy for M_2

($\mathbf{NoEq}_X$), so R is a partial function and injective. By Theorem 2, terms M_2 with parameters W and W' are related by R. Let

$$s = run\, run^f\ run^{f,1}\ q^x\ 1^x\ q^y\ 0^y\ run^{abort}\ done^{abort}\ done^{f,1}\ done^f\ done$$

be a complete play for M_2 with parameter W. Then, the corresponding R-closed subset of $[\![M_2]\!]^{comp}_{W'}$ is:

$$S' = \{run\, run^f\ run^{f,1}\ q^x\ ff^x\ q^y\ tt^y\ run^{abort}\ done^{abort}\ done^{f,1}\ done^f\ done\}\ \square$$

5 Threshold Collections

In this section we present theorems which provide sufficient conditions for reducing the verification of properties for all interpretations of X, to the verification of the same properties for finite interpretations of X.

Theorem 3. *Let $\Gamma \vdash M, N : T$ be terms which satisfy ($\mathbf{NoEq}_X$), $W_0 = \{0\}$, and all data values of X in M and N are from W_0.*
(i) If $\Gamma \vdash_{W_0} M \not\cong N$ then $\Gamma \vdash_W M \not\cong N$ for all W.
(ii) If $\Gamma \vdash_{W_0} M$ is not safe then $\Gamma \vdash_W M$ is not safe for all W.
(iii) If $\Gamma \vdash_{W_0} M$ is safe then $\Gamma \vdash_W M$ is safe for all W.

Proof. Let $R : W \longleftrightarrow W_0$ be the unique total function from W to W_0. By Theorem 2, we have that

$$[\![\Gamma \vdash N]\!]^{comp}_W\ (Str^{comp}_{[\![\Gamma \vdash T]\!]_{R,W,W_0}})\ [\![\Gamma \vdash N]\!]^{comp}_{W_0} \quad (1)$$
$$[\![\Gamma \vdash M]\!]^{comp}_{W_0}\ (Str^{comp}_{[\![\Gamma \vdash T]\!]_{R^{-1},W_0,W}})\ [\![\Gamma \vdash M]\!]^{comp}_{W} \quad (2)$$

We will prove (i) by contraposition. Suppose that $\Gamma \vdash_W M \sqsubseteq_{\approx} N$ for some W. Let $t \in [\![\Gamma \vdash M]\!]^{comp}_{W_0}$. By (2), there exists $t' \in [\![\Gamma \vdash M]\!]^{comp}_{W}$ such that $t(L_{[\![\Gamma \vdash T]\!]_{R^{-1},W_0,W}})t'$ (*). Since $\Gamma \vdash_W M \sqsubseteq_{\approx} N$, we have $t' \in [\![\Gamma \vdash N]\!]^{comp}_{W}$. Now by (1), there exists $t^\dagger \in [\![\Gamma \vdash N]\!]^{comp}_{W_0}$ such that $t'\, L_{[\![\Gamma \vdash T]\!]_{R,W,W_0}}\, t^\dagger$ (**). By (*), (**), the fact that $W_0 \times W_0$ is the identity relation, it follows that $t = t^\dagger$. Therefore, $[\![\Gamma \vdash M]\!]^{comp}_{W_0} \subseteq [\![\Gamma \vdash N]\!]^{comp}_{W_0}$, and so $\Gamma \vdash_{W_0} M \sqsubseteq_{\approx} N$. The cases (ii) and (iii) are similar. $\square$

Theorem 4. *Let $\Gamma \vdash M, N : T$ be terms, κ be a nonzero integer such that $W_\kappa = \{0, \ldots, \kappa\}$, and all data values of X in M and N are from W_κ.*
(i) If $\Gamma \vdash_{W_\kappa} M \not\cong N$ then $\Gamma \vdash_{W_{\kappa'}} M \not\cong N$ for all $\kappa' \geq \kappa$.
(ii) If $\Gamma \vdash_{W_\kappa} M$ is not safe then $\Gamma \vdash_{W_{\kappa'}} M$ is not safe for all $\kappa' \geq \kappa$.

Proof. Consider the case (i). The proof is by contraposition. Suppose that $\Gamma \vdash_{W_{\kappa'}} M \sqsubseteq_{\approx} N$ for some $\kappa' \geq \kappa$. Let $R : W_\kappa \longleftrightarrow W_{\kappa'}$ be a total function and injective. By Theorem 2,

$$[\![\Gamma \vdash M]\!]^{comp}_{W_\kappa}\ (Str^{comp}_{[\![\Gamma \vdash T]\!]_{R,W_\kappa,W_{\kappa'}}})\ [\![\Gamma \vdash M]\!]^{comp}_{W_{\kappa'}} \quad (1)$$

Let $t \in [\![\Gamma \vdash M]\!]^{comp}_{W_\kappa}$. By (1), there exists $t' \in [\![\Gamma \vdash M]\!]^{comp}_{W_{\kappa'}}$ such that $t(L_{[\![\Gamma \vdash T]\!]_{R,W_\kappa,W_{\kappa'}}})t'$ (*). By assumption, we have $t' \in [\![\Gamma \vdash N]\!]^{comp}_{W_{\kappa'}}$. Let $\pi : W_{\kappa'} \longleftrightarrow W_\kappa$ be such that $\pi : W_{\kappa'} \backslash Range(R) \to W_\kappa$ is a total function. Then $R^{-1} \cup \pi : W_{\kappa'} \longleftrightarrow W_\kappa$ is a total function and surjective. But $W_\kappa \subseteq W_{\kappa'}$, $(R^{-1} \cup \pi) \upharpoonright W_\kappa$ is injective, and $t' \in Domain(L_{[\![\Gamma \vdash T]\!]_{(R^{-1} \cup \pi) \upharpoonright W_\kappa, W_{\kappa'}, W_\kappa}})$. In a manner similar to the proof of Theorem 2, we can show that there exists $t^\dagger \in [\![\Gamma \vdash N]\!]^{comp}_{W_\kappa}$ such that $t'(L_{[\![\Gamma \vdash T]\!]_{R^{-1} \cup \pi, W_{\kappa'}, W_\kappa}})t^\dagger$ (**). It follows from (*), (**), and the definition of R, that $t = t^\dagger$. Therefore, $\Gamma \vdash_{W_\kappa} M \sqsubseteq N$.

Consider the case (ii). Suppose that $\Gamma \vdash_{W_{\kappa'}} M$ is safe for some $\kappa' \geq \kappa$. Let $R : W_\kappa \longleftrightarrow W_{\kappa'}$ be a total function and injective. By Theorem 2, $[\![\Gamma \vdash M]\!]^{comp}_{W_\kappa} (Str^{comp}_{[\![\Gamma \vdash T]\!]_{R,W_\kappa,W_{\kappa'}}}) [\![\Gamma \vdash M]\!]^{comp}_{W_{\kappa'}}$ (1). Let $t \in [\![\Gamma \vdash M]\!]^{comp}_{W_\kappa}$. By (1), there exists $t' \in [\![\Gamma \vdash M]\!]^{comp}_{W_{\kappa'}}$ such that $t(L_{[\![\Gamma \vdash T]\!]_{R,W_\kappa,W_{\kappa'}}})t'$ (*). But t' is safe by assumption, i.e. it does not contain unsafe moves. So it must be that t is also safe. □

Example 3. The term M_1 from Example 1 with parameter $W_0 = \{0\}$ is abort-safe. By Theorem 3, we can conclude that M_1 is abort-safe for all W. So M_1 where X is replaced by int is also abort-safe.

The term M_2 from Example 2 with parameter $W_1 = \{0, 1\}$ is abort-unsafe. By Theorem 4, it follows that M_2 is abort-unsafe for all $W_{\kappa'}$, $\kappa' \geq 1$ (which also includes int). □

6 Application

From now on, we restrict the programming language to the 2nd-order recursion-free fragment. More precisely, function types are restricted to $T ::= B \mid B \to T$. This restriction is made since the game semantic model for this language fragment is decidable, i.e. the model can be given concrete automata-theoretic representations using the regular languages as in [6] and the CSP process algebra as in [4], and so a range of verification problems such as approximation and safety can be solved algorithmically. We have extended the tool given in [4], which supports only IA terms, such that it automatically converts a predicatively parametrically polymorphic IA term into a parameterized CSP process [15] which represents its game semantics. The resulting CSP process is defined by a script in machine readable CSP which the tool outputs.

Let us consider an implementation of the linear search algorithm:

$$
\begin{array}{l}
x[k] : \mathsf{var}X,\ y : \mathsf{exp}X,\ \mathsf{abort} : \mathsf{com} \vdash \\
\quad \mathsf{new}_X\ a[k]\ \mathsf{in}\ \mathsf{new}_{int_{k+1}}\ i := 0\ \mathsf{in} \\
\quad \mathsf{while}\,(i < k)\,\mathsf{do}\,\{a[i] := x[i];\ \ i := i + 1;\ \} \\
\quad \mathsf{new}_X\ z := y\ \mathsf{in}\ \mathsf{new}_{bool}\ present := false\ \mathsf{in} \\
\quad \mathsf{while}\,(i < k)\,\mathsf{do}\,\{\ \mathsf{if}\,(\mathbf{a[i]} = \mathbf{z})\,\mathsf{then}\,present := true;\ \ i := i + 1;\ \} \\
\quad \mathsf{if}\,(\neg\, present)\,\mathsf{then}\,\mathsf{abort}\ : \mathsf{com}
\end{array}
$$

The code includes a meta variable $k > 0$, representing array size, which will be replaced by several different values. The data stored in the arrays x, a, and the

expression y is of type X, and the type of index i is int_{k+1}, i.e. one more than the size of the array. The program first copies the input array x into a local array a, and the input expression y into a local variable z. Then, the local array is searched for an occurrence of the value stored in z. If the search fails, then abort is executed. The equality is the only operation on the data of type X (see the bold in the code), so this term does not satisfy ($\mathbf{NoEq}_X$).

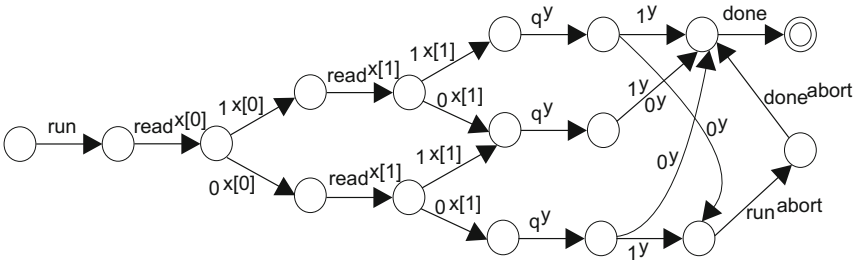

Fig. 3. Model for the linear search with k=2 and $W_1 = \{0, 1\}$.

A model for the term with $k = 2$ and parameter $W_1 = \{0, 1\}$ is shown in Fig. 3. If the value read from y has occurred in x then the term terminates successfully without executing abort; otherwise the term runs abort.

If this term is tested for abort-safety, we obtain the following counter-example:

$$run \;\; read^{x[0]} \;\; 1^{x[0]} \;\; read^{x[1]} \;\; 1^{x[1]} \;\; q^{y} \;\; 0^{y} \;\; run^{abort} \;\; done^{abort} done$$

By Theorem 4, it follows that this term is abort-unsafe for all $W_{\kappa'}$, $\kappa' \geq 1$. So if X is replaced by int, the term is also abort-unsafe. We performed experiments for the linear search term with different sizes of k and $W_\kappa = \{0, \ldots, \kappa\}$, by converting the term into a CSP process and then using FDR model checker [2] to generate its model and test its abort-safety.

Experimental results are shown in Table 1. The execution time is given in seconds, and the size of the final model in number of states. We ran FDR on a Machine AMD Phenom II X4 940 with 4GB RAM. We can see that the model and the time increase very fast as we increase the size of W_κ. Still by using Theorem 4, it only suffices to check the term with parameter W_1 in order to infer its safety for all $W_{\kappa'}$, $\kappa' \geq 1$.

6.1 Integration with Abstraction Refinement

We can combine our parameterized approach with an abstraction refinement procedure (ARP) [3], since both approaches can be applied to terms which contain infinite (integer) types. The former approach will be used to handle all

[2] http://www.fsel.com

Table 1. Verification of linear search

	W_1		W_2		W_3	
k	Time	Model	Time	Model	Time	Model
5	2	36	2	68	2	124
10	7	66	8	138	10	274
15	18	96	20	208	39	424
20	40	126	47	278	160	574

data-independent integer types, which will be treated as parameters, while the rest of infinite types will be handled by the latter approach.

In the ARP (see [3] for details), instead of finite integers we introduce a new data type of abstracted integers int_π. We use the following finitary abstractions π: $[\,] = \{\mathbb{Z}\}$, $[n, m] = \{< n, n, \ldots, 0, \ldots, m-1, m, > m\}$, where Z, $<n = \{n' \mid n' < n\}$, and $>n = \{n' \mid n' > n\}$ are abstract values. Abstractions are refined by splitting abstract values. We check safety of $\Gamma \vdash_W M : T$ (with infinite integer data types) by performing a sequence of iterations. The initial abstracted term $\Gamma_0 \vdash_W M_0 : T_0$ uses the coarsest abstraction $[\,]$ for any integer identifier. In every iteration, the game semantics model of the abstracted term is checked for safety. If no counterexample or a genuine one is found, the procedure terminates. Otherwise, if a spurious counter-example is found, it is used to generate a refined abstracted term, which is passed to the next iteration.

For example, let us reconsider the linear search term. The ARP needs $k + 2$ iterations to automatically adjust the type of the local variable i from the coarsest abstraction with a single abstract value $[\,]$ to the abstraction $[0, k]$. For such abstraction of i ($int_{[0,k]}$), a genuine counter-example is found.

7 Conclusion

The paper presents how we can automatically verify parameterized terms for all instances of the parameter X. We described here the case where there is one data type variable X. If there is more than one such variable, the obtained results can be applied to one at a time. The approach proposed here can be also extended to terms with nondeterminism [5], concurrency [7], and other features.

References

1. Abramsky, S., McCusker, G.: Linearity, sharing and state: a fully abstract game semantics for Idealized Algol with active expressions. In: O'Hearn, P.W., Tennent, R.D. (eds.) Algol-like languages, Birkhaüser, Boston (1997)
2. Abramsky, S., McCusker, G.: Game Semantics. In: Proceedings of the 1997 Marktoberdorf Summer School: Computational Logic, pp. 1–56. Springer, Heidelberg (1998)
3. Dimovski, A., Ghica, D.R., Lazić, R.: Data-Abstraction Refinement: A Game Semantic Approach. In: Hankin, C., Siveroni, I. (eds.) SAS 2005. LNCS, vol. 3672, pp. 102–117. Springer, Heidelberg (2005)

4. Dimovski, A., Lazić, R.: Compositional Software Verification Based on Game Semantics and Process Algebras. Int. Journal on STTT 9(1), 37–51 (2007)
5. Dimovski, A.: A Compositional Method for Deciding Equivalence and Termination of Nondeterministic Programs. In: Méry, D., Merz, S. (eds.) IFM 2010. LNCS, vol. 6396, pp. 121–135. Springer, Heidelberg (2010)
6. Ghica, D.R., McCusker, G.: The Regular-Language Semantics of Second-order Idealized Algol. Theoretical Computer Science 309(1–3), 469–502 (2003)
7. Ghica, D.R., Murawski, A.S., Ong, C.-H.L.: Syntactic control of concurrency. In: Díaz, J., Karhumäki, J., Lepistö, A., Sannella, D. (eds.) ICALP 2004. LNCS, vol. 3142, pp. 683–694. Springer, Heidelberg (2004)
8. Hughes, D.J.D.: Hypergame Semantics: Full Completness for System F. D. Phil. Thesis, Oxford University (1999)
9. Laird, J.: Game Semantics for a Polymorphic Programming Language. In: Proceedings of LICS 2010. IEEE, pp. 41–49. IEEE, Los Alamitos (2010)
10. Lazić, R.: A Semantic Study of Data Independence with Applications to Model Checking. D. Phil. Thesis, Oxford University (1999)
11. Lazić, R., Nowak, D.: A Unifying Approach to Data-Independence. In: Wagner, T.A., Rana, O.F. (eds.) AA-WS 2000. LNCS (LNAI), vol. 1887, pp. 581–595. Springer, Heidelberg (2001)
12. Murawski, A.S., Ouaknine, J.: On Probabilistic Program Equivalence and Refinement. In: Abadi, M., de Alfaro, L. (eds.) CONCUR 2005. LNCS, vol. 3653, pp. 156–170. Springer, Heidelberg (2005)
13. Ma, Q., Reynolds, J.C.: Types, Abstraction, and Parametric Polymorphism, Part 2. In: Schmidt, D., Main, M.G., Melton, A.C., Mislove, M.W., Brookes, S.D. (eds.) MFPS 1991. LNCS, vol. 598, pp. 1–40. Springer, Heidelberg (1992)
14. O'Hearn, P.W., Tennent, R.D.: Parametricity and Local Variables. Journal of the ACM 42(3), 658–709 (1995)
15. Roscoe, W.A.: Theory and Practice of Concurrency. Prentice-Hall, Englewood Cliffs (1998)
16. Wadler, P.: Theorems for Free! In: FPCA 1989, pp. 347–379. ACM, New York (1989)