Ensuring Secure Non-interference of Programs by Game Semantics

Aleksandar S. Dimovski

IT University of Copenhagen, 2300 Copenhagen S, Denmark
adim@itu.dk

Abstract. Non-interference is a security property which states that improper information leakages due to direct and indirect flows have not occurred through executing programs. In this paper we investigate a game semantics based formulation of non-interference that allows to perform a security analysis of closed and open procedural programs. We show that such formulation is amenable to automated verification techniques. The practicality of this method is illustrated by several examples, which also emphasize its advantage compared to known operational methods for reasoning about open programs.

1 Introduction

We address the problem of ensuring secure information flow of programs, which contain two kinds of global variables labeled as: "high security" (secert) and "low-security" (public). Our aim is to prove that a program will not leak sensitive information about its high-security inputs to an external observer (attacker) who can see its low-security outputs. Thus we need to ensure that low-security outputs in all computations of the program do not depend on sensitive high-security inputs. This is also known as *non-interference* property [15], because it states that secret data may not interfere with public data. We can show that the non-interference property holds for a program if no difference in low-security outputs can be observed between any two computations (executions) of the program that differ only on their high-security inputs.

In this paper we propose a game semantics based approach to verify the non-interference property of closed and open programs. Game semantics [1] is a kind of denotational semantics which provides syntax-independent fully abstract (sound and complete) models for a variety of programming languages. This means that the obtained models are sound and complete with respect to observational equivalence of programs, and thus they represent the most accurate models we can find for a programming language. Compared to operational semantics several features of game (denotational) semantics make it very promising for automatic verification of security properties. First, it provides models for any open program fragments, i.e. programs with undefined identifiers such as calls to library functions. This allows us to reason about open programs, which is very difficult to do by using the known operational semantics based techniques. Second, the interpretation of programs is compositional, i.e. it is defined by induction on the program structure, which means that the game semantics model of a larger program is obtained from the models of its constituting subprograms, using a notion of composition. This is essential for achieving modular analysis, which is necessity for scalability when the method is applied to larger programs. Third, game semantics takes into account only extensional properties (what the program computes) of programs. Thus, programs are modeled by how they observationally interact with their environment, and the details of local-state manipulation are hidden, which results in small models with a maximum level of abstraction. This feature of game semantics is very important for efficient establishing of non-interference, since the non-interference also abstracts away from implementation details of a program and focusses on its visible input-output behaviour. Finally, game semantics models have been given certain kinds of concrete automata and process-theoretic representations, and in this way they provide direct support for automatic verification (by model checking) and program analysis of several interesting programming language fragments [9, 6]. Here we present another application of algorithmic game semantics for automatically verifying security properties of programs.

In this work we provide characterizations of non-interference based on the idea of self-composition [2], and on the observation that game semantics is consistent and computationally adequate w.r.t. the operational semantics. In this way the problem of verifying non-interference is reduced to the verification of safety properties of programs. This will enable the use of existing game semantics based verification tools for checking non-interference. If the property does not hold, the tool reports a counter-example trace witnessing insecure computations of the given program.

1.1 Related work

The most common ways in which secret information can be leaked to an external observer are direct and indirect leakages, which are described by the non-interference property. There are also other ways to leak secret information within programs using mechanisms, known as *covert channels* [15], whose primary task is not information transfer. Timing and termination leaks are examples of such covert channels. Here a program can leak sensitive information through its timing (resp., termination) behaviour, where an external attacker can observe the total running time (resp., termination) of programs.

The first attempts to verify security properties were by Denning in [4]. His work on program certification represents an efficient form of static program analysis that can be used to ensure secure information flows. However, this method offers only intuitive arguments for its correctness, and no formal proof is given.

Recently, a new formal approach has been developed to ensure security properties of programs by using security-type systems [11]. In this approach, for every program component are defined security types, which contain information about their types and security levels. Programs that are well-typed under these type systems satisfy certain security properties. Type systems for enforcing non-interference of programs have been proposed by Volpano and Smith in [16], and subsequently they have been extended to detect also covert timing channels in [17]. The main drawback of this approach is its imprecision, since many secure programs are not typable and so are rejected. For example, in the case of non-interference the secure program: $l := h;\ l := 0$, where l and h are low- and high-security variables respectively, is not typable because there is an insecure direct flow in its subprogram $l := h$. One way to address this problem is to use static (information-flow and control-flow) analysis [3]. However, this approach is still imprecise due to the over-approximation and rejects many secure programs.

Semantics based models, that formalize security in terms of program behavior, appear to be essential for allowing more precise security analysis of programs. Security properties can be naturally expressed in terms of end-to-end program behaviour and, thus, become suitable for reasoning by semantic models of programs. For example, in [12, 14] programming-language semantics has been used to rigorously specify and verify the non-interference property.

Game semantics was also previously used for information-flow analysis [13], but based on approximate representations of game semantics models. This was considered only as a possible application of the broader approach to develop program analysis by abstract interpretation in the setting of game semantics. However, this approach did not result in any practical implementations.

2 The Language

Idealized Algol (IA) [1] is a call-by-name λ-calculus extended with imperative features and locally-scoped variables. The data types D are finite integers and booleans ($D ::= \mathsf{int}_n = \{0, \ldots, n-1\} \mid \mathsf{bool} = \{tt, ff\}$). The base types B are expressions, commands, and variables ($B ::= \mathsf{exp}D \mid \mathsf{com} \mid \mathsf{var}D$). In this paper we work with the second-order fragment of IA, denoted as IA_2, where function types T are restricted to have arguments of base types ($T ::= B \mid B \to T$).

Well-typed terms are given by typing judgements of the form $\Gamma \vdash M : T$, where $\Gamma = x_1 : T_1, \ldots, x_k : T_k$ is a type *context* consisting of a finite number of typed free identifiers. Typing rules are standard [1], but the general application rule is broken up into the linear application and the contraction rule [1].

$$\frac{\Gamma \vdash M : B \to T \qquad \Delta \vdash N : B}{\Gamma, \Delta \vdash MN : T} \qquad \frac{\Gamma, x_1 : T, x_2 : T \vdash M : T'}{\Gamma, x : T \vdash M[x/x_1, x/x_2] : T'}$$

We use these two rules to have control over multiple occurrences of free identifiers in terms during typing. The language also contains a set of constants and constructs: $i : \mathsf{expint_n}$ $(0 \leq i < n)$, $tt, ff : \mathsf{expbool}$, $\mathsf{skip} : \mathsf{com}$, $\mathsf{diverge} : \mathsf{com}$, $op : \mathsf{exp}D \to \mathsf{exp}D \to \mathsf{exp}D'$, $; : \mathsf{com} \to B \to B$, $\mathsf{if} : \mathsf{expbool} \to B \to B \to B$, $\mathsf{while} : \mathsf{expbool} \to \mathsf{com} \to \mathsf{com}$, $:= : \mathsf{var}D \to \mathsf{exp}D \to \mathsf{com}$, $! : \mathsf{var}D \to \mathsf{exp}D$, $\mathsf{new_D} : (\mathsf{var}D \to \mathsf{com}) \to \mathsf{com}$, $\mathsf{mkvar_D} : (\mathsf{exp}D \to \mathsf{com}) \to \mathsf{exp}D \to \mathsf{var}D$. Each construct represents a different programming feature, which can also be given in the more traditional form. For example, sequential composition $;\ (M, N) \equiv$

[1] $M[N/x]$ denotes the capture-free substitution of N for x in M.

$M\,;\;N$, branching $\mathsf{if}(M,N_1,N_2) \equiv \mathsf{if}\ M\ \mathsf{then}\ N_1\ \mathsf{else}\ N_2$, assignment $:= (M,N) \equiv M := N$, de-referencing [2] $!(M) \equiv M$, etc. We say that a term M is closed if $\vdash M : T$ is derivable. Any input/output operation in a term is done through global variables (i.e. free identifiers of type $\mathsf{var}D$). So an input is read by de-referencing a global variable, while an output is written by an assignment to a global variable.

A type context is called var-*context* if all identifiers in it have type $\mathsf{var}D$. Given a var-context Γ, we define a Γ-state s as a (partial) function assigning data values to the variables in Γ. We write $St(\Gamma)$ for the set of all such states. Let s be a Γ-state and s' be a Γ'-state such that all variables in Γ and Γ' are distinct. Then, $\mathrm{s} \otimes \mathrm{s}'$ is a $\{\Gamma, \Gamma'\}$-state, such that $\mathrm{s} \otimes \mathrm{s}'(x)$ is equal either to $\mathrm{s}(x)$ if $x \in \Gamma$, or to $\mathrm{s}'(x)$ if $x \in \Gamma'$. The *canonical forms* of the language are defined by $V ::= x \mid v \mid \lambda\, x.M \mid \mathsf{skip} \mid \mathsf{mkvar}_D MN$, where x ranges over identifiers and v ranges over data values of D.

The *operational semantics* is defined by a big-step reduction relation: $\Gamma \vdash M, \mathrm{s} \Longrightarrow V, \mathrm{s}'$, where Γ is a var-context, and s, s' represent Γ-states before and after evaluating the (well-typed) term $\Gamma \vdash M : T$ to a canonical form V. Reduction rules are standard (see [1] for details). Since the language is deterministic, every term can be reduced to at most one canonical form.

Given a term $\Gamma \vdash M : \mathsf{com}$, where Γ is a var-context, we say that M *terminates* in state s, written $M, \mathrm{s} \Downarrow$, if $\Gamma \vdash M, \mathrm{s} \Longrightarrow \mathsf{skip}, \mathrm{s}'$ for some state s'. If M is a closed term then we abbreviate the relation $M, \emptyset \Downarrow$ with $M \Downarrow$. We define a *program context* $C[-] \in Ctxt(\Gamma, T)$ to be a term with (several occurrences of) a hole in it, such that if $\Gamma \vdash M : T$ is derivable then $\vdash C[M] : \mathsf{com}$. We say that a term $\Gamma \vdash M : T$ is an *approximate* of a term $\Gamma \vdash N : T$, denoted by $\Gamma \vdash M \sqsubseteq N$, if and only if for all program contexts $C[-] \in Ctxt(\Gamma, T)$, if $C[M] \Downarrow$ then $C[N] \Downarrow$. If two terms approximate each other they are considered *observationally-equivalent*, denoted by $\Gamma \vdash M \cong N$.

Let $\Gamma, \Delta \vdash M : T$ be a term where Γ is a *var*-context and Δ is an arbitrary context. Such terms are called *split terms*, and we denote them as $\Gamma \mid \Delta \vdash M : T$. If Δ is empty, then these terms are called *semi-closed*. The semi-closed terms have only some global variables, and the operational semantics is defined only for them. In the following we fix a context $\Gamma_1 = l : \mathsf{var}D, h : \mathsf{var}D'$, where l represents a low-security (public) variable and h represents a high-security (secret) variable. We say that h is *non-interfering* with l in $\Gamma_1 \mid - \vdash M : \mathsf{com}$ if the same values for l and different values for h in a state prior to evaluation of the term M always result in a state after the evaluation of M where values for l are the same.

Definition 1. *A variable h is* non-interfering *with l in $\Gamma_1 \mid - \vdash M : \mathsf{com}$ if*

$$\begin{aligned} \forall\, s_1, s_2 \in St(\Gamma_1).\ & s_1(l) = s_2(l) \wedge s_1(h) \neq s_2(h) \wedge \\ & \Gamma_1 \vdash M, \mathrm{s}_1 \Longrightarrow \mathsf{skip}, \mathrm{s_1}' \wedge \Gamma_1 \vdash M, \mathrm{s}_2 \Longrightarrow \mathsf{skip}, \mathrm{s_2}' \\ & \Rightarrow s_1'(l) = s_2'(l) \end{aligned} \qquad (1)$$

[2] De-referencing is made explicit in the syntax. Thus, $!M$ is a term of type $\mathsf{exp}D$ denoting the contents of a term M of type $\mathsf{var}D$.

As it was argued in [2], the formula (1), where two evaluations (computations) of the same term are considered, can be replaced by an equivalent formula, where we consider only one evaluation of the sequential composition of the given term with another its copy, such that the global variables are suitably renamed in the latter. So sequential composition enables us to place these two evaluations one after the other. Let $\Gamma_1 \vdash M : \mathsf{com}$ be a term, we define $\Gamma_1' = l' : \mathsf{var}D, h' : \mathsf{var}D'$, and $M' = M[l'/l, h'/h]$. We will use these definitions for Γ_1' and M' in the rest of the paper. The following can be shown: $\Gamma_1 \vdash M, \mathrm{s}_1 \Longrightarrow \mathsf{skip}, \mathrm{s}_1{}' \wedge \Gamma_1' \vdash M', \mathrm{s}_2 \Longrightarrow \mathsf{skip}, \mathrm{s}_2{}'$ iff $\Gamma_1, \Gamma_1' \vdash M;\ M', \mathrm{s}_1 \otimes \mathrm{s}_2 \Longrightarrow \mathsf{skip}, \mathrm{s}_1{}' \otimes \mathrm{s}_2{}'$. In this way, we provide an alternative definition to formula (1) as follows. We say that a variable h is *non-interfering* with l in a semi-closed term $\Gamma_1 \mid - \vdash M : \mathsf{com}$ if

$$\begin{aligned} \forall\, s_1 \in St(\Gamma_1), s_2 \in St(\Gamma_1').\ & s_1(l) = s_2(l') \wedge s_1(h) \neq s_2(h') \wedge \\ & \Gamma_1, \Gamma_1' \vdash M;\ M', \mathrm{s}_1 \otimes \mathrm{s}_2 \Longrightarrow \mathsf{skip}, \mathrm{s}_1{}' \otimes \mathrm{s}_2{}' \\ & \Rightarrow s_1'(l) = s_2'(l') \end{aligned} \qquad (2)$$

It is easy to show that (1) and (2) are equivalent.

Definition 2. *We say that a variable h is* non-interfering *with l in a* split (open) term $\Gamma_1 \mid \Delta \vdash M : \mathsf{com}$*, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$, if for all program contexts $C[-] \in Ctxt(\Gamma, T)$ that do not contain any occurrences of variables h and l, we have that h is non-interfering with l in $\Gamma_1 \mid - \vdash C[M] : \mathsf{com}$.*

Finally, we say that a term is *secure* if all global high-security variables are non-interfering with any of its global low-security variables.

3 Game Semantics

The game semantics model for IA$_2$ can be represented by regular languages [9], which we sketch below. In game semantics, a kind of game is played by two participants: the Player, who represents the term being modeled, and the Opponent, who represents the context (environment) in which the term is used. The two alternate to make *moves*, which can be either a question (a demand for information) or an answer (a supply of information). Types are interpreted as *arenas* in which games are played, computations as *plays* of a game, and terms as *strategies* (sets of plays) for a game. In the regular-language representation, arenas (types) are expressed as *alphabets of moves*, plays (computations) as *words*, and strategies (terms) as *regular-languages* over alphabets of moves.

Each type T is interpreted by an alphabet of moves $\mathcal{A}_{[\![T]\!]}$, which can be partitioned into two subsets of *questions* $Q_{[\![T]\!]}$ and *answers* $A_{[\![T]\!]}$.

$$Q_{[\![\mathsf{exp}D]\!]} = \{q\} \qquad A_{[\![\mathsf{exp}D]\!]} = D$$
$$Q_{[\![\mathsf{com}]\!]} = \{run\} \qquad A_{[\![\mathsf{com}]\!]} = \{done\}$$
$$Q_{[\![\mathsf{var}D]\!]} = \{read, write(a) \mid a \in D\} \qquad A_{[\![\mathsf{var}D]\!]} = D \cup \{ok\}$$

For function types, we have $\mathcal{A}_{[\![B_1^1 \to \ldots \to B_k^k \to B]\!]} = \sum_{1 \leq i \leq k} \mathcal{A}^i_{[\![B_i]\!]} + \mathcal{A}_{[\![B]\!]}$, where $+$ means a disjoint union of alphabets. We will use superscript tags to keep record from which type of the disjoint union each move originates. Each move in an alphabet represents an observable action that a term of the corresponding type can perform. So for expressions, we have a question move q to request the value of the expression, and possible responses are taken from the data type D. For commands, there is a question move *run* to initiate a command, and an answer move *done* to signal successful termination of a command. For variables, we have moves for writing to the variable: $\mathit{write}(a)$, which is acknowledged by the move *ok*, and for reading from the variable: a question move *read*, and answers are from D.

Terms in β-normal form are interpreted by regular languages specified by *extended regular expressions* R. They are defined inductively over finite alphabets $\mathcal{A}$ using the following operations: the empty language $\emptyset$, the empty word ε, the elements of the alphabet $a \in \mathcal{A}$, concatenation $R \cdot S$, Kleene star R^*, union $R + S$, intersection $R \cap S$, restriction $R \mid_{\mathcal{A}'}$ $(\mathcal{A}' \subseteq \mathcal{A})$ which removes from words of R all letters from $\mathcal{A}'$, substitution $R[S/w]$ which replaces all occurrences of the subword w in words of R by words of S, composition $R \fatsemi S$ which is defined below, and shuffle $R \bowtie S$ which generates the set of all possible interleavings $w_1 \bowtie w_2$ for any words w_1 of R and w_2 of S. Composition of regular expressions R defined over alphabet $\mathcal{A}^1 + \mathcal{B}^2$ and S over $\mathcal{B}^2 + \mathcal{C}^3$ is given as follows:

$$R \fatsemi_{\mathcal{B}^2} S = \{ w\big[s^1/a^2 \cdot b^2\big] \ \mid \ w \in S, a^2 \cdot s^1 \cdot b^2 \in R \}$$

where R is a set of words of the form $a^2 \cdot s^1 \cdot b^2$, such that a^2, $b^2 \in \mathcal{B}^2$ and s contains only letters from $\mathcal{A}^1$. So the composition is defined over $\mathcal{A}^1 + \mathcal{C}^3$, and all letters of $\mathcal{B}^2$ are removed. It is a standard result that any extended regular expression obtained from the operations above denotes a regular language [9, pp. 11–12], which can be recognized by a finite automaton.

A term $\mathit{\Gamma} \vdash M : T$ is interpreted by a regular expression $\llbracket \mathit{\Gamma} \vdash M : T \rrbracket$ defined over the alphabet

$$\mathcal{A}_{\llbracket \mathit{\Gamma} \vdash T \rrbracket} = \big(\sum_{x : T' \in \mathit{\Gamma}} \mathcal{A}^x_{\llbracket T' \rrbracket} \big) + \mathcal{A}_{\llbracket T \rrbracket}$$

where all moves corresponding to types of free identifiers are tagged with the names of those free identifiers. Every word in $\llbracket \mathit{\Gamma} \vdash M : T \rrbracket$ corresponds to a complete play in the strategy for $\mathit{\Gamma} \vdash M : T$, and it represents the observable effects of a completed computation of the term.

Free identifiers are interpreted by the so-called copy-cat (identity) strategies, which contain all possible computations that terms of that type can have. In this way they provide the most general context in which an open term can be used. The general definition is:

$$\begin{aligned} &\llbracket \mathit{\Gamma}, x : B_1^{x,1} \to \ldots B_k^{x,k} \to B^x \vdash x : B_1^1 \to \ldots B_k^k \to B \rrbracket = \\ &\qquad \sum_{q \in Q_{\llbracket B \rrbracket}} q \cdot q^x \cdot \Big(\sum_{1 \leq i \leq k} \big(\sum_{q_1 \in Q_{\llbracket B_i \rrbracket}} q_1^{x,i} \cdot q_1^i \cdot \sum_{a_1 \in A_{\llbracket B_i \rrbracket}} a_1^i \cdot a_1^{x,i} \big) \Big)^* \cdot \sum_{a \in A_{\llbracket B \rrbracket}} a^x \cdot a \end{aligned}$$

So if a first-order non-local function x is called, it may evaluate any of its arguments, zero or more times, in an arbitrary order and then it can return any allowable answer from $A_{[\![B]\!]}$ as a result. For example, $[\![\mathit{\Gamma}, x : \mathsf{exp}D \vdash x : \mathsf{exp}D]\!] = q \cdot q^x \cdot \sum_{n \in D} n^x \cdot n$. Here Opponent starts any play (word) by asking what is the value of this expression with the move q, and Player responds by asking what is the value of the non-local expression x with the move q^x. Then Opponent can provide an arbitrary value n from D for x, which will be copied by Player as answer to the first question q.

For the linear application, we have $[\![\mathit{\Gamma}, \mathit{\Delta} \vdash M\,N : T]\!] = [\![\mathit{\Delta} \vdash N : B^1]\!] \,{}_{9}^{\circ}\mathcal{A}^1_{[\![B]\!]}$ $[\![\mathit{\Gamma} \vdash M : B^1 \to T]\!]$. The contraction $[\![\mathit{\Gamma}, x : T^x \vdash M[x/x_1, x/x_2] : T']\!]$ is obtained from $[\![\mathit{\Gamma}, x_1 : T^{x_1}, x_2 : T^{x_2} \vdash M : T']\!]$, such that the moves associated with x_1 and x_2 are de-tagged so that they represent actions associated with x.

To represent local variables, we first need to define a (storage) 'cell' strategy cell_v which imposes the good variable behaviour on the local variable. So cell_v responds to each $\mathit{write}(n)$ with ok, and plays the most recently written value in response to read, or if no value has been written yet then answers the read with the initial value v. Then we have:

$$\mathsf{cell}_v = (\mathit{read} \cdot v)^* \cdot \Big(\sum_{n \in D} \mathit{write}(n) \cdot \mathit{ok} \cdot (\mathit{read} \cdot n)^*\Big)^*$$

$$[\![\mathit{\Gamma} \vdash \mathsf{new}_D\, x := v \,\mathsf{in}\, M : B]\!] = \big([\![\mathit{\Gamma}, x : \mathsf{var}D \vdash M]\!] \cap (\mathsf{cell}^x_v \bowtie \mathcal{A}^*_{[\![\mathit{\Gamma} \vdash B]\!]})\big) \mid_{\mathcal{A}^x_{[\![\mathsf{var}D]\!]}}$$

Note that all actions associated with x are hidden away in the final model for new, since x is a local variable and so not visible outside of the term.

Language constants and constructs are interpreted as follows:

$$[\![\mathsf{v} : \mathsf{exp}D]\!] = \{q \cdot v\} \qquad [\![\mathsf{skip} : \mathsf{com}]\!] = \{\mathit{run} \cdot \mathit{done}\} \qquad [\![\mathsf{diverge} : \mathsf{com}]\!] = \emptyset$$
$$[\![\mathsf{op} : \mathsf{exp}D^1 \times \mathsf{exp}D^2 \to \mathsf{exp}D']\!] = q \cdot q^1 \cdot \textstyle\sum_{m \in D} m^1 \cdot q^2 \cdot \sum_{n \in D} n^2 \cdot (m\; \mathit{op}\; n)$$
$$[\![;\ : \mathsf{com}^1 \to \mathsf{com}^2 \to \mathsf{com}]\!] = \mathit{run} \cdot \mathit{run}^1 \cdot \mathit{done}^1 \cdot \mathit{run}^2 \cdot \mathit{done}^2 \cdot \mathit{done}$$
$$[\![\mathsf{if} : \mathsf{expbool}^1 \to \mathsf{com}^2 \to \mathsf{com}^3 \to \mathsf{com}]\!] = \mathit{run} \cdot q^1 \cdot \mathit{tt}^1 \cdot \mathit{run}^2 \cdot \mathit{done}^2 \cdot \mathit{done} + \mathit{run} \cdot q^1 \cdot \mathit{ff}^1 \cdot \mathit{run}^3 \cdot \mathit{done}^3 \cdot \mathit{done}$$
$$[\![\mathsf{while} : \mathsf{expbool}^1 \to \mathsf{com}^2 \to \mathsf{com}]\!] = \mathit{run} \cdot (q^1 \cdot \mathit{tt}^1 \cdot \mathit{run}^2 \cdot \mathit{done}^2)^* \cdot q^1 \cdot \mathit{ff}^1 \cdot \mathit{done}$$
$$[\![:=: \mathsf{var}D^1 \to \mathsf{exp}D^2 \to \mathsf{com}]\!] = \textstyle\sum_{n \in D} \mathit{run} \cdot q^2 \cdot n^2 \cdot \mathit{write}(n)^1 \cdot \mathit{ok}^1 \cdot \mathit{done}$$
$$[\![! : \mathsf{var}D^1 \to \mathsf{exp}D]\!] = \textstyle\sum_{n \in D} q \cdot \mathit{read}^1 \cdot n^1 \cdot n$$

We can see that any constant v is modeled by a word where the initial question q is answered by the value of that constant, whereas the "do-nothing" command skip is modeled by a word where Player immediately responds to the initial question run with done. The regular expression for any arithmetic-logic operation op asks for values of the arguments with moves q^1 and q^2, and after obtaining them by m and n responds to the initial question q by the value $(m\, \mathsf{op}\, n)$.

Example 1. Consider the term:

$$n : \mathsf{exp\,int_2}^n,\, c : \mathsf{com}^c \vdash \mathsf{new}_{\mathsf{int}_2}\, x := 0 \,\mathsf{in\,if}\, (!x = n) \,\mathsf{then}\, c \,\mathsf{else\,skip} : \mathsf{com}$$

The model of this term is: $\mathit{run} \cdot q^n \cdot (0^n \cdot \mathit{run}^c \cdot \mathit{done}^c + 1^n) \cdot \mathit{done}$.

In the model are represented observable interactions of the term with its environment, so we can see only the moves associated with the non-local identifiers

n and c as well as with the top-level type com. When the term (Player) asks for the value of n with the move q^n, the environment (Opponent) provides an answer which can be 0 or 1, because the data type of n is $\mathsf{int}_2 = \{0, 1\}$. If the value 0 is provided for n, then since x has also initial value 0 the command c is executed by moves run^c and $done^c$. Otherwise, if 1 is provided for n, the term terminates without running c. Note that all moves associated with the local variable x are not visible in the final model. □

3.1 Formal Properties

We first show how this model is related with the operational semantics. In order to do this, the state needs to be represented explicitly in the model. A Γ-state s, where $\Gamma = x_1 : \mathsf{var}D_1, \ldots, x_k : \mathsf{var}D_k$, is interpreted by the following strategy:

$$[\![\mathrm{s} : \mathsf{var}D_1^{x_1} \times \ldots \times \mathsf{var}D_k^{x_k}]\!] = \mathsf{cell}^{x_1}_{\mathrm{s}(x_1)} \bowtie \ldots \bowtie \mathsf{cell}^{x_k}_{\mathrm{s}(x_k)}$$

So $[\![s]\!]$ is defined over the alphabet $\mathcal{A}^{x_1}_{[\![\mathsf{var}D_1]\!]} + \ldots + \mathcal{A}^{x_k}_{[\![\mathsf{var}D_k]\!]}$, and words in $[\![s]\!]$ are such that projections onto x_i-component are the same as those of suitable initialized $\mathsf{cell}_{\mathrm{s}(x_i)}$ strategies. The interpretation of $\Gamma \vdash M : \mathsf{com}$ at state s is:

$$[\![\Gamma \vdash M]\!] \circ [\![\mathrm{s}]\!] = \big([\![\Gamma \vdash M]\!] \cap ([\![\mathrm{s}]\!] \bowtie \mathcal{A}^*_{[\![\mathsf{com}]\!]})\big) \mid_{\mathcal{A}_{[\![\Gamma]\!]}}$$

which is defined over the alphabet $\mathcal{A}_{[\![\mathsf{com}]\!]}$. This interpretation can be studied more closely by considering the words in which moves associated with Γ are not hidden. Such words are called *interaction sequences*. For any interaction sequence $run \cdot t \cdot done$ from $[\![\Gamma \vdash M]\!] \circ [\![\mathrm{s}]\!]$, where t is an even-length word over $\mathcal{A}_{[\![\Gamma]\!]}$, we say that it leaves the state s' if the last write moves in each x_i-component are such that x_i is set to the value $\mathrm{s}'(x_i)$. For example, let $\mathrm{s} = (x \mapsto 1, y \mapsto 2)$, then the interaction sequence, $run \cdot write(5)^y \cdot ok^y \cdot read^x \cdot 1^x \cdot done$, leaves the state $\mathrm{s}' = (x \mapsto 1, y \mapsto 5)$. The following results are proved in [1] for the full IA, but they also hold for the fragment we use here.

Lemma 1. *If $\Gamma \vdash M : \{\mathsf{com}, \mathsf{expD}\}$ and $\Gamma \vdash M, s \Longrightarrow V, s'$, then for each interaction sequence $i \cdot t$ from $[\![\Gamma \vdash V]\!] \circ [\![s']\!]$ (i is an initial move) we have some interaction $i \cdot t' \cdot t$ from $[\![\Gamma \vdash M]\!] \circ [\![s]\!]$ such that t' is a word over $\mathcal{A}_{[\![\Gamma]\!]}$ which leaves the state s'. Moreover, every interaction sequence from $[\![\Gamma \vdash M]\!] \circ [\![s]\!]$ is of this form.*

Theorem 1 (Consistency). *If $\Gamma \vdash M, s \Longrightarrow V, s'$ then $[\![\Gamma \vdash M]\!] \circ [\![s]\!] = [\![\Gamma \vdash V]\!] \circ [\![s']\!]$.*

Theorem 2 (Computational Adequacy). *If $[\![\Gamma \vdash M]\!] \circ [\![s]\!] = run \cdot done$ then $\Gamma \vdash M, s \Longrightarrow \mathsf{skip}, s'$.*

Theorem 3 (Full Abstraction). *$\Gamma \vdash M \cong N$ iff $[\![\Gamma \vdash M]\!] = [\![\Gamma \vdash N]\!]$.*

Suppose that there is a special free identifier abort of type com^{abort} in Γ. We say that a term $\Gamma \vdash M$ is *safe* iff $\Gamma \vdash M[\mathsf{skip}/\mathsf{abort}] \underset{\sim}{\sqsubset} M[\mathsf{diverge}/\mathsf{abort}]$; otherwise we say that a term is *unsafe*. Since the game-semantics model is fully abstract, the following can be shown (see also [5]).

Lemma 2. *A term $\mathit{\Gamma} \vdash M$ is safe iff $[\![\mathit{\Gamma} \vdash M]\!]$ does not contain any play with moves from $\mathcal{A}^{abort}_{[\![\mathsf{com}]\!]}$, which we call unsafe plays.*

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

4 Checking Non-interference

We first describe how the game semantics model can be used to check the non-interference property of a semi-closed term.

Theorem 4. *Let $\mathit{\Gamma}_1 \mid - \vdash M : \mathsf{com}$ be a semi-closed term. We have that* [3] [4]

$$\begin{aligned}[\![k : \mathsf{exp}D, k' : \mathsf{exp}D', \mathsf{abort} : \mathsf{com} \vdash\ & \mathsf{new}_D\ l := k\ \mathsf{in}\ \mathsf{new}_{D'}\ h := k'\ \mathsf{in} \\ & \mathsf{new}_D\ l' := !l\ \mathsf{in}\ \mathsf{new}_{D'}\ h' := k'\ \mathsf{in} \\ & M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort} : \mathsf{com}]\!]\end{aligned} \tag{3}$$

contains no unsafe plays Iff *h is non-interfering with l in $\mathit{\Gamma}_1 \mid - \vdash M : \mathsf{com}$ as defined by (2).*

Proof. Let us assume that the term in (3) is safe. Let $\mathit{\Delta} = k : \mathsf{exp}D, k' : \mathsf{exp}D'$, then we have:

$$\begin{aligned}&[\![\mathit{\Delta}, \mathsf{abort} : \mathsf{com} \vdash \mathsf{new}_D\ l := k\ \mathsf{in}\ \mathsf{new}_{D'}\ h := k'\ \mathsf{in}\ \mathsf{new}_D\ l' := !l\ \mathsf{in}\ \mathsf{new}_{D'}\ h' := k'\ \mathsf{in} \\ &\qquad\qquad M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort}]\!] = \\ &[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1', \mathsf{abort} : \mathsf{com} \vdash M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort}]\!] \circ [\![\mathrm{s}_1 \otimes \mathrm{s}_2]\!]\end{aligned}$$

where $\mathrm{s}_1 = (l \mapsto v_1, h \mapsto v_2)$ and $\mathrm{s}_2 = (l' \mapsto v_1, h' \mapsto v_2')$, for arbitrary values $v_1 \in D$, $v_2, v_2' \in D'$. By assumption $[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1', \mathsf{abort} : \mathsf{com} \vdash M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort}]\!] \circ [\![\mathrm{s}_1 \otimes \mathrm{s}_2]\!]$ is safe, so any of its interaction sequences leaves the state $\mathrm{s}_1' \otimes \mathrm{s}_2'$, such that $\mathrm{s}_1'(l) = \mathrm{s}_2'(l')$. Otherwise, we would have unsafe plays. The last if statement does not change the state, because it does not contain write moves. So by Theorem 2 and Lemma 1 it follows that the fact (2) holds.

For the opposite direction, we assume that the fact (2) holds. Let consider $[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1', \mathsf{abort} : \mathsf{com} \vdash M;\ M']\!] \circ [\![\mathrm{s}_1 \otimes \mathrm{s}_2]\!]$, where $\mathrm{s}_1 = (l \mapsto v_1, h \mapsto v_2)$ and $\mathrm{s}_2 = (l' \mapsto v_1, h' \mapsto v_2')$, for arbitrary values $v_1 \in D$, $v_2, v_2' \in D'$. By Theorem 1 and Lemma 1, any interaction sequence in $[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1', \mathsf{abort} : \mathsf{com} \vdash M;\ M']\!] \circ [\![\mathrm{s}_1 \otimes \mathrm{s}_2]\!]$ leaves the state $\mathrm{s}_1' \otimes \mathrm{s}_2'$, such that $\mathrm{s}_1'(l) = \mathrm{s}_2'(l')$. Therefore the last if statement in $[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1', \mathsf{abort} : \mathsf{com} \vdash M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort}]\!] \circ [\![\mathrm{s}_1 \otimes \mathrm{s}_2]\!]$ is evaluated in the state $\mathrm{s}_1' \otimes \mathrm{s}_2'$, and so its condition always evaluates to false, which implies that this model has no unsafe plays. Thus, the term in (3) is safe. □

[3] We use the free identifier k in (3) to initialize the variables l and l' to an arbitrary value from D which is the same for both l and l', while k' is used to initialize the variables h and h' to arbitrary (possibly different) values from D'.

[4] After declaring local variables in the term in (3), we can add the command: $\mathsf{if}\ (!h = !h')\ \mathsf{then\ diverge}$, in order to eliminate from the model all redundant computations for which initial values of h and h' are the same.

We can verify the non-interference property of a semi-closed term by checking safety of the term extended as in the formula (3). If its model is safe, then the term does satisfy the non-interference property; otherwise a counter-example (unsafe play) is reported, which shows how high-security information can be leaked by the term.

Example 2. Consider the term from the Introduction section:

$$l, h : \mathsf{var\,int}_2 \vdash l := !h : \mathsf{com}$$

To verify that h is non-interfering with l, we need to check the safety of the following term obtained by (3):

$$\begin{aligned} k, k' : \mathsf{expint}_2, \mathsf{abort} : \mathsf{com} \vdash\ & \mathsf{new}_{int_2}\ l := k\ \mathsf{in}\ \mathsf{new}_{int_2}\ h := k'\ \mathsf{in} \\ & \mathsf{new}_{\mathsf{int}_2}\ l' := !l\ \mathsf{in}\ \mathsf{new}_{\mathsf{int}_2}\ h' := k'\ \mathsf{in} \\ & l := !h;\ l' := !h';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort} : \mathsf{com} \end{aligned}$$

The game semantics model of this term contains all possible observable interactions of the term with its environment, which contains non-local identifiers k, k', and abort. The model is represented by the following regular expression:

$$\begin{aligned} & run \cdot q^k \cdot (0^k + 1^k) \cdot q^{k'} \cdot \big(0^{k'} \cdot q^{k'} \cdot 0^{k'} + 1^{k'} \cdot q^{k'} \cdot 1^{k'} + \\ & \qquad (0^{k'} \cdot q^{k'} \cdot 1^{k'} + 1^{k'} \cdot q^{k'} \cdot 0^{k'}) \cdot run^{abort} \cdot done^{abort}\big) \cdot done \end{aligned}$$

where the value for k read from the environment is used to initialize l and l', and the two values read for k' are used to initialize h and h' respectively.

We can see that this model contains four unsafe plays corresponding to all possible combinations of initial values (from $\mathsf{int}_2 = \{0, 1\}$) for l, h, l', and h', such that the values for l and l' are the same and the values for h and h' are different. For example, an unsafe play is:

$$run \cdot q^k \cdot 0^k \cdot q^{k'} \cdot 0^{k'} \cdot q^{k'} \cdot 1^{k'} \cdot run^{abort} \cdot done^{abort} \cdot done$$

corresponding to two computations of the given term with initial values of h: 0 and 1 respectively, which will have two different final values for l. The interaction sequence corresponding to this unsafe play, where all interactions with local variables l, h, l', and h' are not hidden, is the following:

$$\begin{aligned} & run \cdot q^k \cdot 0^k \cdot write(0)^l \cdot ok^l \cdot q^{k'} \cdot 0^{k'} \cdot write(0)^h \cdot ok^h \cdot read^l \cdot 0^l \cdot write(0)^{l'} \cdot ok^{l'} \cdot \\ & \quad q^{k'} \cdot 1^{k'} \cdot write(1)^{h'} \cdot ok^{h'} \cdot read^h \cdot 0^h \cdot write(0)^l \cdot ok^l \cdot read^{h'} \cdot 1^{h'} \cdot \\ & \qquad write(1)^{l'} \cdot ok^{l'} \cdot read^l \cdot 0^l \cdot read^{l'} \cdot 1^{l'} \cdot run^{abort} \cdot done^{abort} \cdot done \end{aligned}$$

It becomes apparent from this interaction sequence that the different initial values for h and h' are propagated as final values for l and l' respectively, which in effect causes the abort to be executed.

Let us check the security of the term:

$$l, h : \mathsf{var\,int}_2 \vdash l := !h;\ l := 0 : \mathsf{com}$$

The model of the term extended by using (3) is given by:

$$run \cdot q^{k} \cdot (0^{k} + 1^{k}) \cdot q^{k'} \cdot (0^{k'} + 1^{k'}) \cdot q^{k'} \cdot (0^{k'} + 1^{k'}) \cdot done$$

This model contains no unsafe plays, and so we can conclude that the corresponding term is secure. □

To verify non-interference and security of a split (open) term $\Gamma_1 \mid \Delta \vdash M : \mathsf{com}$, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$, we need to check the state after evaluating

$$\Gamma_1, \Gamma_1' \vdash C[M]; C[M'], \mathrm{s}_1 \otimes \mathrm{s}_2 \tag{4}$$

for all contexts $C[-]$ that do not contain any occurrences of variables from Γ_1 and Γ_1', and for all states $\mathrm{s}_1 \in St(\Gamma_1)$, $\mathrm{s}_2 \in St(\Gamma_1')$, such that $\mathrm{s}_1(l) = \mathrm{s}_2(l')$. We can decompose the term $C[M]$ as follows:

$$C[M] = \big(\lambda\, u : T_1 \times \ldots \times T_k \to \mathsf{com}\,.C[u]\big)(\lambda\, x_1 : T_1. \ldots .\lambda\, x_k : T_k.M)$$

and its game semantics is: $[\![\Gamma_1 \vdash C[M]]\!] = [\![\Gamma_1, \Delta \vdash M]\!] \,\text{⨾}\, [\![u \vdash C[u]]\!]$. Since variables from Γ_1 and Γ_1' do not occur in $C[-]$, the state $\mathrm{s}_1 \otimes \mathrm{s}_2$ in the term in (4) can be changed only when terms M or M' are evaluated in the context $C[-]$. As for the case of semi-closed terms, we can check the values of l and l' in the state left after evaluating the term $M; M'$. But now terms M and M' are run in the same context $C[-]$, so we are interested in examining only those behaviors of $M;\, M'$ in which free identifiers from Δ behave uniformly in M and M'. If we remove these additional constraints, we obtain an *over-approximated model* $[\![\Gamma_1, \Gamma_1', \Delta \vdash M; M']\!]$ in which M and M' are run in possibly different contexts, i.e. all identifiers from Δ can behave freely in both M and M'.

Theorem 5. *Let* $\Gamma_1 \mid \Delta \vdash M : \mathsf{com}$ *be a split (open) term, and* $\Delta = x_1 : T_1, \ldots, x_k : T_k$*. If the model*

$$\begin{aligned}[\![k : \mathsf{exp}D, k' : \mathsf{exp}D', \mathsf{abort} : \mathsf{com}, \Delta \vdash\ & \mathsf{new}_D\ l := k \ \mathsf{in}\ \mathsf{new}_{D'}\ h := k' \ \mathsf{in} \\ & \mathsf{new}_D\ l' := !l \ \mathsf{in}\ \mathsf{new}_{D'}\ h' := k' \ \mathsf{in} \\ & M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort} : \mathsf{com}]\!]\end{aligned} \tag{5}$$

contains no unsafe plays, Then h *is non-interfering with* l *in* $\Gamma_1 \mid \Delta \vdash M : \mathsf{com}$.

Note that if an unsafe play is found in (5), it does not follow that M is insecure, i.e. the found counter-example may be spurious introduced due to the over-approximation in (5). This is the case, since free identifiers of type T are modeled by copy-cat strategies, which contain all possible behaviours of terms of type T. So by using game semantics we generate the most general context for the term $M;\ M'$ in (5), but we would like the obtained context for M to be the same for M', because we use the term $M; M'$ only to compare two different computations of the same term M.

Example 3. Consider the term:

$$l, h : \mathsf{varint}_2, f : \mathsf{com}^{f,1} \to \mathsf{com}^{f} \vdash f(l := 1) : \mathsf{com}$$

where f is a non-local call-by-name function. The model for this term is:

$$run \cdot run^f \cdot (run^{f,1} \cdot write(1)^l \cdot ok^l \cdot done^{f,1})^* \cdot done^f \cdot done$$

It represents all possible computations of the term, i.e. f may evaluate its argument, zero or more times, and then the term terminates successfully. Notice that moves tagged with f represent the actions of calling and returning from the function f, while moves tagged with $f, 1$ are the actions caused by evaluating the first argument of f. We can see that whenever f calls its argument, the value 1 is written into l.

We can check that the model corresponding to the extended term obtained by (5) contains the unsafe play:

$$\begin{aligned} run \cdot q^k \cdot 0^k \cdot q^{k'} \cdot 1^{k'} \cdot q^{k'} \cdot 0^{k'} \cdot run^f \cdot run^{f,1} \cdot done^{f,1} \cdot done^f \cdot \\ run^f \cdot done^f \cdot run^{abort} \cdot done^{abort} \cdot done \end{aligned}$$

It shows two computations with the initial value of l set to 0, where the first one corresponds to evaluating f which calls its argument once, and the second corresponds to evaluating f which does not call its argument at all. Both will have two different final values for l: 1 and 0 respectively. But this represents a spurious counter-example, since f does not behave uniformly in the two computations, i.e. it calls its argument in the first but not in the second computation. □

In order to address the above problem, we define an *under-approximation* of the required model, which is a regular language and can be used for deciding insecurity of split terms. Suppose that $\mathit{\Gamma}_1 \mid \mathit{\Delta} \vdash M$ is derived without using the contraction rule for $\mathit{\Delta}$, i.e. any identifier from $\mathit{\Delta}$ occurs at most once in M. We define a model which runs M and M' in the same context as follows:

$$\begin{aligned} [\![\mathit{\Gamma}_1, \mathit{\Gamma}_1' \mid \mathit{\Delta} \vdash M;\ M']\!]^m = [\![\mathit{\Gamma}_1, \mathit{\Gamma}_1' \mid \mathit{\Delta} \vdash M;\ M']\!] \cap \\ (\mathsf{delta}^{x_1}_{T_1,m} \bowtie \ldots \bowtie \mathsf{delta}^{x_k}_{T_k,m} \bowtie \mathcal{A}^*_{[\![\mathit{\Gamma}_1, \mathit{\Gamma}_1' \vdash \mathsf{com}]\!]}) \end{aligned} \qquad (6)$$

where $m \geq 0$ denotes the number of times that identifiers from $\mathit{\Delta}$ of first-order function type may evaluate their arguments, and $\mathsf{delta}_{T,m}$ runs an arbitrary behaviour of type T zero, once, or two times. It is defined inductively on types T as follows.

$$\begin{aligned} \mathsf{delta}_{\mathsf{exp}D,m} &= \epsilon + q \cdot \textstyle\sum_{n \in D} n \cdot (\epsilon + q \cdot n) \\ \mathsf{delta}_{\mathsf{com},m} &= \epsilon + run \cdot done \cdot (\epsilon + run \cdot done) \\ \mathsf{delta}_{\mathsf{var}D,m} &= \epsilon + (read \cdot \textstyle\sum_{n \in D} n \cdot (\epsilon + read \cdot n)) + \\ &\qquad (\textstyle\sum_{n \in D} write(n) \cdot ok \cdot (\epsilon + write(n) \cdot ok)) \end{aligned}$$

for any $m \geq 0$. In the case of first-order function types T, in order $\mathsf{delta}_{T,m}$ to be a regular language, we have to limit the number of times its arguments can be evaluated. For simplicity, we will only give the definition for $\mathsf{com} \to \mathsf{com}$ whose argument can be evaluated at most m times.

$$\mathsf{delta}_{\mathsf{com}^1 \to \mathsf{com},m} = \epsilon + run \cdot \sum_{r=0}^{m} (run^1 \cdot done^1)^r \cdot done \cdot (\epsilon + run \cdot (run^1 \cdot done^1)^r \cdot done)$$

If $T = B_1 \to \ldots \to B_k \to B$, i.e. it has k arguments, we have to remember not only how many times arguments are evaluated in the first call, but also the exact order in which arguments are evaluated.

Let some identifier from Δ occur more than once in M. Let $\Gamma_1 \mid \Delta_1 \vdash M_1$ be derived without using the contraction for Δ_1, such that $\Gamma_1 \mid \Delta \vdash M$ is obtained from it by applying one or more times the contraction rule for identifiers from Δ. In this case, $[\![\Gamma_1, \Gamma_1' \mid \Delta \vdash M;\ M']\!]^m$ is generated by first computing $[\![\Gamma_1, \Gamma_1' \mid \Delta_1 \vdash M_1;\ M_1']\!]^m$ as described above, and then by suitable tagging all moves associated with several occurrences of the same identifier from Δ. So we have that $[\![\Gamma_1, \Gamma_1', \Delta \vdash M;\ M']\!]^m$, for any $m \geq 0$, is an under-approximation of the required model where M and M' are run in the same context. Thus, we can use it to check (in)security of split terms.

Theorem 6. *Let $\Gamma_1 \,|\, \Delta \vdash M$ be a split (open) term, and $\Delta = x_1 : T_1, \ldots, x_k : T_k$. If the model*

$$\begin{aligned}[\![k : \mathsf{exp}D, k' : \mathsf{exp}D', \mathsf{abort} : \mathsf{com}, \Delta \vdash\ & \mathsf{new}_D\ l := k \ \mathsf{in}\ \mathsf{new}_{D'}\ h := k' \ \mathsf{in} \\ & \mathsf{new}_D\ l' := !l \ \mathsf{in}\ \mathsf{new}_{D'}\ h' := k' \ \mathsf{in} \\ & M;\ M';\ \mathsf{if}\ (!l \neq !l')\ \mathsf{then\ abort} : \mathsf{com}]\!]^m\end{aligned} \tag{7}$$

is unsafe (contains unsafe plays), Then $\Gamma_1 \mid \Delta \vdash M$ does not satisfy the non-interference property between l and h.

In the above definition of $\mathsf{delta}_{T,m}$ we allow an arbitrary behavior of type T to be played zero, once, or two times, since it is possible that a term does not evaluate an occurrence of a free identifier. In our case, this means that it is possible an occurrence of a free identifier from Δ to be evaluated only by M, or only by M', or by none of them.

Example 4. Consider the term:

$$\Gamma_1 \mid z, w : \mathsf{exp\,int}_2 \vdash \mathsf{if}\ (!h > 0)\ \mathsf{then}\ l := z\ \mathsf{else}\ l := w : \mathsf{com}$$

This term is not secure, and the counter-example witnessing this contains one computation where l is updated by z, and another one where l is updated by w. So this counter-example will be captured by the model defined in (7), only if $\mathsf{delta}_{T,m}$ is defined as above, i.e. it may run exactly once a behaviour of T. □

We can combine results in Theorems 5 and 6 to obtain a procedure for verifying the non-interference between l and h in $\Gamma_1 \,|\, \Delta \vdash M$ as follows. First, we generate the model in (7) for some $m > 0$ and check its safety. If an unsafe play is found, then it witnesses that the non-interference property is not satisfied. Otherwise, if the model in (7) is safe, we generate the model in (5) and check its safety. If no unsafe plays are found, then h is non-interfering with l in M.

5 Applications

The game semantics model presented here can be also given concrete representation by using the CSP process algebra [6]. The verification tool in [6] automatically converts an IA term into a CSP process that represents its game semantics.

Safety of terms is then verified by calls to the FDR tool, which is a model checker for the CSP process algebra. In the input syntax, we use simple type annotations to indicate what finite sets of integers will be used to model free identifiers and local variables of type integer. An operation between values of types int_{n_1} and int_{n_2} produces a value of type $\mathsf{int}_{max\{n_1,n_2\}}$. The operation is performed modulo $max\{n_1, n_2\}$. The tool in [6] was used to practically check the security of the following terms.

We first analyse an introductory (semi-closed) term M_1:

$$\begin{aligned} l, h : \mathsf{varint}_3 \vdash\ & \mathsf{new}_{\mathsf{int}_3}\, x := 0 \;\mathsf{in}\; \mathsf{new}_{\mathsf{int}_3}\, y :=!h \;\mathsf{in} \\ & \mathsf{while}\,(!x <!y)\;\mathsf{do}\; x :=!x + 1; \\ & \mathsf{if}\;(!x > 0)\;\mathsf{then}\; l := 1 : \mathsf{com} \end{aligned}$$

whose model is given by: $run \cdot read^h \cdot \left(0^h + (1^h + 2^h) \cdot write(1)^l \cdot ok^l\right) \cdot done.$

We can see that if the value of h read from the environment is 0 then the guards of while and if commands are both false causing the term to terminate immediately. Otherwise, if the value of h is 1 or 2, then the body of while where x is increased will be run once or two times respectively, which makes the guard of if command to evaluate to true and subsequently the value 1 is written into l.

If the term M_1 is extended by using formula (3), we obtain a counter-example (unsafe play) corresponding to two computations with initial values of l set to 0 and the initial value of h set to 0 in the first and to 1 (or 2) in the second computation.

Let us consider the term M_2 defined as:

$$l, h : \mathsf{varint}_2, f : \mathsf{com}^{f,1} \to \mathsf{com}^f \vdash f(\mathsf{if}\;(!h \neq 0)\;\mathsf{then}\; l := 1) : \mathsf{com}$$

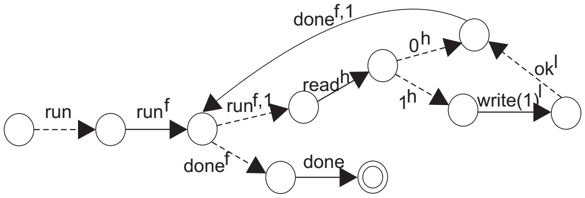

Fig. 1. The model for M_2

The model representing this term is shown in Fig. 1. When f calls its argument, the term asks for the value of h by $read^h$. If the value provided from the environment is different from 0, the value 1 is written into l.

Insecurity of this term can be established by checking safety of the extended term obtained by the formula (7) for any $m \geq 1$. For example, if $m = 1$ the following genuine unsafe play is found:

$$\begin{aligned} & run \cdot q^k \cdot 0^k \cdot q^{k'} \cdot 1^{k'} \cdot q^{k'} \cdot 0^{k'} \cdot run^f \cdot run^{f,1} \cdot done^{f,1} \cdot done^f \cdot \\ & \qquad run^f \cdot run^{f,1} \cdot done^{f,1} \cdot done^f \cdot run^{abort} \cdot done^{abort} \cdot done \end{aligned}$$

This unsafe play corresponds to one computation where l is set to 0, h is set to 1, and the argument of f is evaluated once; and another computation where initial values of l and h are both 0, and f also calls its argument once. The final value of l will be 1 in the first case and 0 in the second.

Consider the term M_3 which implements the linear-search algorithm:

```
$l, h : \mathsf{varint}_2, x[k] \ : \ \mathsf{varint}_2 \vdash$
    $\mathsf{new}_{int_{k+1}}\ i := 0 \ \mathsf{in} \ \mathsf{new}_{int_2}\ y := !h \ \mathsf{in}$
    $\mathsf{new}_{bool}\ present := ff \ \mathsf{in}$
    $\mathsf{while}\ (i < k)\ \mathsf{do}\ \{\ \mathsf{if}\ (!x[i] = !y)\ \mathsf{then}\ present := tt; \quad i := !i + 1; \ \}$
    $\mathsf{if}\ (\neg\, !present)\ \mathsf{then}\ l := 1 \ \ : \mathsf{com}$
```

The meta variable $k > 0$ represents the array size. The term first copies the input value of h into a local variable y. The linear-search algorithm is then used to find whether the value stored in y is in the array x. If the the value is not found, l is updated to 1.

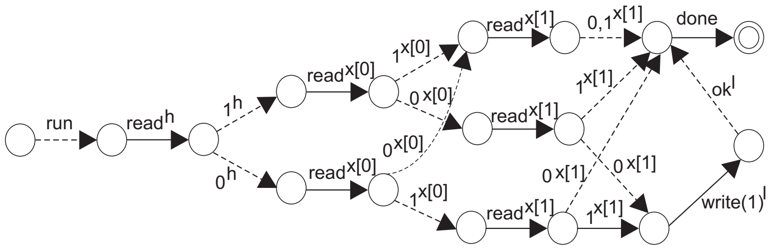

Fig. 2. The model for M_3 with k=2

In Fig. 2 is shown the model for this term with $k = 2$. If the value read from the environment for h is not present in any element of the array x, then the value 1 is written into l. Otherwise, the term terminates without writing into l. We can analyse this term with different values of k and different finite sets of integers used to model global variables l, h, and the array x, by generating the model in (7) for $m = 0$. We obtain that this term is insecure, with a counter-example corresponding to two computations, such that initial values of h are different, the initial value of l is 0, and the search succeeds in the one and fails in the other computation, thus they are leaving two different final values for l.

6 Conclusion

We presented a game semantics based approach for verifying security properties of closed and open sequential programs. The applicability of this approach was illustrated with several examples.

This work also has the potential to be applied to problems such as security of terms with infinite data types and verifying various types of security properties.

If we want to verify security of terms with infinite data types, such as integers, we can use some of the existing methods based on game semantics for verifying safety of such terms, such as counter-example guided abstraction refinement procedure (ARP) [5] or symbolic representation of game semantics [8]. In [7], game semantics based approach is used to verify some other security properties, such as timing and termination leaks. To detect such leaks, slot-game semantics [10] for a quantitative analysis of programs is used.

References

1. Abramsky, S., and McCusker, G: Game Semantics. In Proceedings of *the 1997 Marktoberdorf Summer School: Computational Logic* , (1998), 1–56. Springer.
2. Barthe, G., D'Argenio, P.R., Rezk, T: Secure information flow by self-composition. In: IEEE CSFW 2004. pp. 100–114. IEEE Computer Society Press, (2004).
3. Clark, D., Hankin, C., and Hunt, S: Information flow for Alogol-like languages. In *Computer Languages* **28(1)**, pp. 3–28, (2002).
4. Denning, D.E: *Cryptography and Data Security.* Addison-Wesley, Reading, MA, 1982.
5. 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).
6. Dimovski, A., Lazić, R: Compositional Software Verification Based on Game Semantics and Process Algebras. In *Int. Journal on STTT* **9(1)**, pp. 37–51, (2007).
7. Dimovski, A: Slot Games for Detecting Timing Leaks of Programs. In: Puppis, G., Villa, T. (eds.) GandALF 2013. EPTCS vol. 119, pp. 166–179. Open Publishing Association, (2013).
8. Dimovski, A: Program Verification Using Symbolic Game Semantics. In *Theoretical Computer Science (TCS)*, (01/2014).
9. Ghica, D. R., McCusker, G: The Regular-Language Semantics of Second-order Idealized Algol. Theoretical Computer Science **309** (1–3), pp. 469–502, (2003).
10. Ghica, D. R. Slot Games: a quantitative model of computation. In Palsberg, J., Abadi, M. (eds.) POPL 2005. ACM, pp. 85–97. ACM Press, New York (1998).
11. Heintze, N., Riecke, J.G: The SLam calculus: programming with secrecy and integrity. In: MacQueen, D.B., Cardelli, L. (eds.) POPL 1998. ACM, pp. 365–377. ACM, New York (1998).
12. Joshi, R., and Leino, K.R.M: A semantic approach to secure information flow. In *Science of Computer Programming* **37**, pp. 113–138, (2000).
13. Malacaria, M., and Hankin, C: Non-deterministic games and program analysis: An application to security. In: LICS 1999, pp. 443–452. IEEE Computer Society Press, Los Alamitos (1999).
14. McLean, J: Proving noninterference and functional correctness using traces. In: *J. Computer Security*, **vol. 1, no. 1**, pp. 3758, (1992).
15. Sabelfeld, A., and Myers, A.C: Language-based information-flow security. In IEEE *Journal on Selected Areas in Communications* **21(1)**, (2003), 5–19.
16. Volpano, D., Smith, G., and Irvine, C: A sound type system for secure flow analysis. In *Journal of Computer Security* **4(2/3)**, (1996), 167–188.
17. Volpano, D., Smith, G: Eliminating covert flows with minimum typings. In: IEEE *Computer Security Foundations Workshop (CSFW),* 1997, 156–169. IEEE Computer Society Press, (1997).