Slot Games for Detecting Timing Leaks of Programs

Aleksandar S. Dimovski
Faculty of Information-Communication Tech., FON University, Skopje, 1000, MKD
aleksandar.dimovski@fon.edu.mk

In this paper we describe a method for verifying secure information flow of programs, where apart from direct and indirect flows a secret information can be leaked through covert timing channels. That is, no two computations of a program that differ only on high-security inputs can be distinguished by low-security outputs and timing differences. We attack this problem by using slot-game semantics for a quantitative analysis of programs. We show how slot-games model can be used for performing a precise security analysis of programs, that takes into account both extensional and intensional properties of programs. The practicality of this approach for automated verification is also shown.

1 Introduction

Secure information flow analysis is a technique which performs a static analysis of a program with the goal of proving that it will not leak any sensitive (secret) information improperly. If the program passes the test, then we say that it is secure and can be run safely. There are several ways in which secret information can be leaked to an external observer. The most common are direct and indirect leakages, which are described by the so-called non-interference property [13, 18]. We say that a program satisfies the non-interference property if its high-security (secret) inputs do not affect its low-security (public) outputs, which can be seen by external observers.

However, a program can also leak information through its timing behaviour, where an external observer can measure its total running time. Such timing leaks are difficult to detect and prevent, because they can exploit low-level implementation details. To detect timing leaks, we need to ensure that the total running time of a program do not depend on its high-security inputs.

In this paper we describe a game semantics based approach for performing a precise security analysis. We have already shown in [8] how game semantics can be applied for verifying the non-interference property. Now we use slot-game semantics to check for timing leaks of closed and open programs. We focus here only on detecting covert timing channels, since the non-interference property can be verified similarly as in [8]. Slot-game semantics was developed in [11] for a quantitative analysis of Algol-like programs. It is suitable for verifying the above security properties, since it takes into account both extensional (*what* the program computes) and intensional (*how* the program computes) properties of programs. It represents a kind of denotational semantics induced by the theory of operational improvement of Sands [19]. Improvement is a refinement of the standard theory of operational approximation, where we say that one program is an improvement of another if its execution is more efficient in any program context. We will measure efficiency of a program as the sum of costs associated with basic operations it can perform. It has been shown that slot-game semantics is fully abstract (sound and complete) with respect to operational improvement, so we can use it as a denotational theory of improvement to analyse programming languages.

The advantages of game semantics (denotational) based approach for verifying security are several. We can reason about open programs, i.e. programs with non-locally defined identifiers. Moreover, game semantics is compositional, which enables analysis about program fragments to be combined into an

To appear in EPTCS.

analysis of a larger program. Also the model hides the details of local-state manipulation of a program, which results in small models with maximum level of abstraction where are represented only visible input-output behaviours enriched with costs that measure their efficiency. All other behaviour is abstracted away, which makes this model very suitable for security analysis. Finally, the game model for some language fragments admits finitary representation by using regular languages or CSP processes [10, 6], and has already been applied to automatic program verification. Here we present another application of algorithmic game semantics for automatically verifying security properties of programs.

Related work. The most common approach to ensure security properties of programs is by using security-type systems [14]. Here 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 [20], and subsequently they have been extended to detect also covert timing channels in [21, 2]. A drawback of this approach is its imprecision, since many secure programs are not typable and so are rejected. A more precise analysis of programs can be achieved by using semantics-based approaches [15].

2 Syntax and Operational Semantics

We will define a secure information flow analysis for Idealized Algol (IA), a small Algol-like language introduced by Reynolds [16] which has been used as a metalanguage in the denotational semantics community. It is a call-by-name λ-calculus extended with imperative features and locally-scoped variables. In order to be able to perform an automata-theoretic analysis of the language, we consider here its second-order recursion-free fragment (IA_2 for short). It contains finitary data types D: $\mathsf{int}_n = \{0,\ldots,n-1\}$ and $\mathsf{bool} = \{tt, ff\}$, and first-order function types: $T ::= B \mid B \to T$, where B ranges over base types: expressions ($\mathsf{exp}D$), commands (com), and variables ($\mathsf{var}D$).

Syntax of the language is given by the following grammar:

$$\begin{array}{c} M ::= x \mid v \mid \mathsf{skip} \mid \mathsf{diverge} \mid M\,\mathsf{op}\,M \mid M;M \mid \mathsf{if}\,M\,\mathsf{then}\,M\,\mathsf{else}\,M \mid \mathsf{while}\,M\,\mathsf{do}\,M \\ \mid M := M \mid !M \mid \mathsf{new}_D\,x := v\,\mathsf{in}\,M \mid \mathsf{mkvar}_D MM \mid \lambda\, x.M \mid MM \end{array}$$

where v ranges over constants of type D.

Typing judgements are of the form $\Gamma \vdash M : T$, where Γ is a type *context* consisting of a finite number of typed free identifiers. Typing rules of the language 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.

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.

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

$\Gamma \vdash n_1 \,\mathsf{op}\, n_2, \mathrm{s} \longrightarrow^{k_{op}} n, \mathrm{s},\ \text{where}\, n = n_1 \mathrm{op} n_2$
$\Gamma \vdash \mathsf{skip}; \mathsf{skip}, \mathrm{s} \longrightarrow^{k_{seq}} \mathsf{skip}, \mathrm{s}'$
$\Gamma \vdash \mathsf{if}\, tt\, \mathsf{then}\, M_1\, \mathsf{else} M_2, \mathrm{s} \longrightarrow^{k_{if}} M_1, \mathrm{s}$
$\Gamma \vdash \mathsf{if}\, ff\, \mathsf{then}\, M_1\, \mathsf{else} M_2, \mathrm{s} \longrightarrow^{k_{if}} M_2, \mathrm{s}$
$\Gamma \vdash x := v', \mathrm{s} \otimes (x \mapsto v) \longrightarrow^{k_{asg}} \mathsf{skip}, \mathrm{s} \otimes (x \mapsto v')$
$\Gamma \vdash !x, \mathrm{s} \otimes (x \mapsto v) \longrightarrow^{k_{der}} v, \mathrm{s} \otimes (x \mapsto v)$
$\Gamma \vdash (\lambda\, x.M)M', \mathrm{s} \longrightarrow^{k_{app}} M[M'/x], \mathrm{s}$
$\Gamma \vdash \mathsf{new}_D\, x := v\, \mathsf{in}\, \mathsf{skip}, \mathrm{s} \longrightarrow^{k_{new}} \mathsf{skip}, \mathrm{s}$

Table 1: Basic Reduction Rules

The operational semantics is defined in terms of a small-step evaluation relation using a notion of an evaluation context [9]. A small-step evaluation (reduction) relation is of the form:

$$\Gamma \vdash M, \mathrm{s} \longrightarrow M', \mathrm{s}'$$

where Γ is a so-called var-context which contains only identifiers of type $\mathsf{var}D$; s, s$'$ are Γ-states which assign data values to the variables in Γ; and M, M' are terms. The set of all Γ-states will be denoted by $St(\Gamma)$.

Evaluation contexts are contexts [2] containing a single hole which is used to identify the next sub-term to be evaluated (reduced). They are defined inductively by the following grammar:

$$E ::= [-] \mid EM \mid E; M \mid \mathsf{skip}; E \mid E \,\mathsf{op}\, M \mid v \,\mathsf{op}\, E \mid \mathsf{if}\, E\, \mathsf{then}\, M\, \mathsf{else}\, M \mid M := E \mid E := v \mid !E$$

The operational semantics is defined in two stages. First, a set of basic reduction rules are defined in Table 1. We assign different (non-negative) costs to each reduction rule, in order to denote how much computational time is needed for a reduction to complete. They are only descriptions of time and we can give them different interpretations describing how much real time they denote. Such an interpretation can be arbitrarily complex. So the semantics is parameterized on the interpretation of costs. Notice that we write $\mathrm{s} \otimes (x \mapsto v)$ to denote a $\{\Gamma, x\}$-state which properly extends s by mapping x to the value v.

We also have reduction rules for iteration, local variables, and mkvar_D construct, which do not incur additional costs.

$$\Gamma \vdash \mathsf{while}\, b\, \mathsf{do}\, M, \mathrm{s} \longrightarrow \mathsf{if}\, b\, \mathsf{then}\, (M; \mathsf{while}\, b\, \mathsf{do}\, M)\, \mathsf{else}\, \mathsf{skip}, \mathrm{s}$$

$$\frac{\Gamma, y \vdash M[y/x], \mathrm{s} \otimes (y \mapsto v) \longrightarrow M', \mathrm{s}' \otimes (y \mapsto v')}{\Gamma \vdash \mathsf{new}_D\, x := v\, \mathsf{in}\, M, \mathrm{s} \longrightarrow \mathsf{new}_D\, x := v'\, \mathsf{in}\, M'[x/y], \mathrm{s}'}$$

$$\Gamma \vdash (\mathsf{mkvar}_D\, M_1 M_2) := v, \mathrm{s} \longrightarrow M_1 v, \mathrm{s} \qquad \Gamma \vdash !(\mathsf{mkvar}_D\, M_1 M_2), \mathrm{s} \longrightarrow M_2, \mathrm{s}$$

Next, the in-context reduction rules for arbitrary terms are defined as:

$$\frac{\Gamma \vdash M, \mathrm{s} \longrightarrow^n M', \mathrm{s}'}{\Gamma \vdash E[M], \mathrm{s} \longrightarrow^n E[M'], \mathrm{s}'}$$

The small-step evaluation relation is deterministic, since arbitrary term can be uniquely partitioned into an evaluation context and a sub-term, which is next to be reduced.

We define the reflexive and transitive closure of the small-step reduction relation as follows:

[2] A context $C[-]$ is a term with (several occurrences of) a hole 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. $\vdash C[M] : \mathsf{com}$.

$$\frac{\Gamma \vdash M, \mathrm{s} \longrightarrow^{n} M', \mathrm{s}'}{\Gamma \vdash M, \mathrm{s} \rightsquigarrow^{n} M', \mathrm{s}'} \qquad \frac{\Gamma \vdash M, \mathrm{s} \rightsquigarrow^{n} M', \mathrm{s}' \qquad \Gamma \vdash M', \mathrm{s}' \rightsquigarrow^{n'} M'', \mathrm{s}''}{\Gamma \vdash M, \mathrm{s} \rightsquigarrow^{n+n'} M'', \mathrm{s}''}$$

Now a theory of operational improvement is defined [19]. Let $\Gamma \vdash M : \mathsf{com}$ be a term, where Γ is a var-context. We say that M *terminates in n steps* at state s, written $M, \mathrm{s} \Downarrow^{n}$, if $\Gamma \vdash M, \mathrm{s} \rightsquigarrow^{n} \mathsf{skip}, \mathrm{s}'$ for some state s'. If M is a closed term and $M, \emptyset \Downarrow^{n}$, then we write $M \Downarrow^{n}$. If $M \Downarrow^{n}$ and $n \leq n'$, we write $M \Downarrow^{\leq n'}$. We say that a term $\Gamma \vdash M : T$ may be *improved* by $\Gamma \vdash N : T$, denoted by $\Gamma \vdash M \gtrsim N$, if and only if for all contexts $C[-]$, if $C[M] \Downarrow^{n}$ then $C[N] \Downarrow^{\leq n}$. If two terms improve each other they are considered *improvment-equivalent*, denoted by $\Gamma \vdash M \approx N$.

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. We say that a semi-closed term $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ does not have *timing leaks* if the initial value of the high-security variable h does not influence the number of reduction steps of M. More formally, we have:

Definition 1. *A semi-closed term* $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ *has no* timing leaks *if*

$$\begin{aligned} \forall s_1, s_2 \in St(\{h\}). \quad & s_1(h) \neq s_2(h) \wedge \\ & h : \mathsf{varD} \vdash M, \mathrm{s}_1 \rightsquigarrow^{n_1} \mathsf{skip}, \mathrm{s}_1{}' \wedge h : \mathsf{varD} \vdash M, \mathrm{s}_2 \rightsquigarrow^{n_2} \mathsf{skip}, \mathrm{s}_2{}' \\ & \Rightarrow n_1 = n_2 \end{aligned} \tag{1}$$

Definition 2. *We say that a* split term $h : \mathsf{varD} \mid \Delta \vdash M : \mathsf{com}$ *does not have timing leaks, where* $\Delta = x_1 : T_1, \ldots, x_k : T_k$*, if for all closed terms* $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$*, we have that the term* $h : \mathsf{varD} \mid - \vdash M[N_1/x_1, \ldots, N_k/x_k] : \mathsf{com}$ *does not have timing leaks.*

The formula (1) can be replaced by an equivalent formula, where instead of two evaluations of the same term we can consider only one evaluation of the sequential composition of the given term with another its copy [3]. So sequential composition enables us to place these two evaluations one after the other. Let $h : \mathsf{varD} \vdash M : \mathsf{com}$ be a term, we define M' to be α-equivalent to $M[h'/h]$ where all bound variables are suitable renamed. The following can be shown: $h \vdash M, \mathrm{s}_1 \rightsquigarrow^{n} \mathsf{skip}, \mathrm{s}_1{}' \wedge h' \vdash M', \mathrm{s}_2 \rightsquigarrow^{n'} \mathsf{skip}, \mathrm{s}_2{}'$ iff $h, h' \vdash M;\ M', \mathrm{s}_1 \otimes \mathrm{s}_2 \rightsquigarrow^{n+n'} \mathsf{skip};\ \mathsf{skip}, \mathrm{s}_1{}' \otimes \mathrm{s}_2{}'$. In this way, we provide an alternative definition to formula (1) as follows. We say that a semi-closed term $h \mid - \vdash M : T$ has no *timing leaks* if

$$\begin{aligned} \forall s_1 \in St(\{h\}), s_2 \in St(\{h'\}). \quad & s_1(h) \neq s_2(h') \wedge \\ & h, h' \vdash M;\ M', \mathrm{s}_1 \otimes \mathrm{s}_2 \rightsquigarrow^{n_1} \mathsf{skip};\ M', \mathrm{s}_1{}' \otimes \mathrm{s}_2 \rightsquigarrow^{n_2} \mathsf{skip};\ \mathsf{skip}, \mathrm{s}_1{}' \otimes \mathrm{s}_2{}' \\ & \Rightarrow n_1 = n_2 \end{aligned} \tag{2}$$

3 Algorithmic Slot-Game Semantics

We now show how slot-game semantics for IA_2 can be represented algorithmically by regular-languages. In this approach, types are interpreted as games, which have two participants: the Player representing the term, and the Opponent representing its context. A game (arena) is defined by means of a set of moves, each being either a question move or an answer move. Each move represents an observable action that a term of a given type can perform. Apart from moves, another kind of action, called *token* (slot), is used to take account of quantitative aspects of terms. It represents a payment that a participant needs to pay in order to use a resource such as time. A computation is interpreted as a play-with-costs, which is given as a sequence of moves and token-actions played by two participants in turns.

We will work here with complete plays-with-costs which represent the observable effects along with incurred costs of a completed computation. Then a term is modelled by a strategy-with-costs, which is a set of complete plays-with-costs. In the regular-language representation of game semantics [10], types (arenas) are expressed as *alphabets of moves*, computations (plays-with-costs) as *words*, and terms (strategies-with-costs) as *regular-languages* over alphabets.

Each type T is interpreted by an alphabet of moves $\mathscr{A}_{[\![T]\!]}$, which can be partitioned into two subsets of *questions* $Q_{[\![T]\!]}$ and *answers* $A_{[\![T]\!]}$. For expressions, we have: $Q_{[\![\mathsf{exp}D]\!]} = \{q\}$ and $A_{[\![\mathsf{exp}D]\!]} = D$, i.e. there are a question move q to ask for the value of the expression and values from D are possible answers. For commands, we have: $Q_{[\![\mathsf{com}]\!]} = \{run\}$ and $A_{[\![\mathsf{com}]\!]} = \{done\}$, i.e. there are a question move *run* to initiate a command and an answer move *done* to signal successful termination of a command. For variables, we have: $Q_{[\![\mathsf{var}D]\!]} = \{read, write(a) \mid a \in D\}$ and $A_{[\![\mathsf{var}D]\!]} = D \cup \{ok\}$, i.e. there are moves for writing to the variable, $write(a)$, acknowledged by the move ok, and for reading from the variable, we have a question move *read*, and an answer to it can be any value from D. For function types, we have $\mathscr{A}_{[\![B^1_1 \to \ldots \to B^k_k \to B]\!]} = \sum_{1 \le i \le k} \mathscr{A}^i_{[\![B_i]\!]} + \mathscr{A}_{[\![B]\!]}$, where $+$ means a disjoint union of alphabets. We will use superscript tags to keep record from which type of the disjoint union each move comes from. We denote the token-action by Ⓢ. A sequence of n token-actions Ⓢ will be written as ⓝ.

For any (β-normal) term we define a regular language specified by an *extended regular expression* R. Apart from the standard operations for generating regular expressions, we will use some more specific operations. We define composition of regular expressions R defined over alphabet $\mathscr{A}^1 + \mathscr{B}^2 + \{Ⓢ\}$ and S over $\mathscr{B}^2 + \mathscr{C}^3 + \{Ⓢ\}$ as follows:

$$R \circ_{9,\mathscr{B}^2} S = \{w[s/a^2 \cdot b^2] \mid w \in S, a^2 \cdot s \cdot b^2 \in R\}$$

where R is a set of words of the form $a^2 \cdot s \cdot b^2$, such that $a^2, b^2 \in \mathscr{B}^2$ and s contains only letters from $\mathscr{A}^1$ and $\{Ⓢ\}$. Notice that the composition is defined over $\mathscr{A}^1 + \mathscr{C}^3 + \{Ⓢ\}$, and all letters of $\mathscr{B}^2$ are hidden. The shuffle operation $R \bowtie S$ generates the set of all possible interleavings from words of R and S, and the restriction operation $R|_{\mathscr{A}'}$ (R defined over $\mathscr{A}$ and $\mathscr{A}' \subseteq \mathscr{A}$) removes from words of R all letters from $\mathscr{A}'$.

If w, w' are words, m is a move, and R is a regular expression, define $m \cdot w \frown w' = m \cdot w' \cdot w$, and $R \frown w' = \{w \frown w' \mid w \in R\}$. Given a word with costs w defined over $\mathscr{A} + \{Ⓢ\}$, we define the underlying word of w as $w^\dagger = w|_{\{Ⓢ\}}$, and the cost of w as $w|_{\mathscr{A}} = ⓝ$, which we denote as $|w| = n$.

The regular expression for $\Gamma \vdash M : T$ is denoted $[\![\Gamma \vdash M : T]\!]$ and is defined over the alphabet $\mathscr{A}_{[\![\Gamma \vdash T]\!]} = \left(\sum_{x:T' \in \Gamma} \mathscr{A}^x_{[\![T']\!]}\right) + \mathscr{A}_{[\![T]\!]} + \{Ⓢ\}$. Every word in $[\![\Gamma \vdash M : T]\!]$ corresponds to a complete play-with-costs in the strategy-with-costs for $\Gamma \vdash M : T$.

Free identifiers $x \in \Gamma$ are interpreted by the copy-cat regular expressions, which contain all possible computations that terms of that type can have. Thus they provide the most general closure of an open term.

$$[\![\Gamma, x : B^{x,1}_1 \to \ldots B^{x,k}_k \to B^x \vdash x : B^1_1 \to \ldots B^k_k \to B]\!] = \sum_{q \in Q_{[\![B]\!]}} q \cdot q^x \cdot \Big(\sum_{1 \le i \le k} \Big(\sum_{q_1 \in Q_{[\![B_i]\!]}} q^{x,i}_1 \cdot q^i_1 \cdot \sum_{a_1 \in A_{[\![B_i]\!]}} a^i_1 \cdot a^{x,i}_1 \Big) \Big)^* \cdot \sum_{a \in A_{[\![B]\!]}} a^x \cdot a$$

When a first-order non-local function is called, it may evaluate any of its arguments, zero or more times, and then it can return any value from its result type as an answer. For example, the term $[\![\Gamma, x : \mathsf{exp}D^x \vdash x : \mathsf{exp}D]\!]$ is modelled by the regular expression: $q \cdot q^x \cdot \sum_{n \in D} n^x \cdot n$.

The linear application is defined as:

$$[\![\Gamma, \Delta \vdash MN : T]\!] = [\![\Delta \vdash N : B^1]\!] \circ_{9,\mathscr{A}^1_{[\![B]\!]}} [\![\Gamma \vdash M : B^1 \to T]\!]$$

Since we work with terms in β-normal form, function application can occur only when the function term is a free identifier. In this case, the interpretation is the same as above except that we add the cost k_{app} corresponding to function application. Notice that k_{app} denotes certain number of Ⓢ units that are needed for a function application to take place. The contraction $[\![\Gamma, x : T^x \vdash M[x/x_1, x/x_2] : T']\!]$ is obtained from $[\![\Gamma, x_1 : T^{x_1}, x_2 : T^{x_2} \vdash M : T']\!]$, 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' regular expression cell_v which imposes the good variable behaviour on the local variable. So cell_v responds to each $write(n)$ 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:

$$\mathsf{cell}_v = (read \cdot v)^* \cdot \Big(\sum_{n \in D} write(n) \cdot ok \cdot (read \cdot n)^* \Big)^*$$
$$[\![\Gamma, x : \mathsf{var}D \vdash M]\!] \circ \mathsf{cell}_v^x = \big([\![\Gamma, x : \mathsf{var}D \vdash M]\!] \cap (\mathsf{cell}_v^x \bowtie (\mathscr{A}_{[\![\Gamma \vdash B]\!]} + Ⓢ)^*) \big) \mid_{\mathscr{A}^x_{[\![\mathsf{var}D]\!]}}$$
$$[\![\Gamma \vdash \mathsf{new}_D x := v \,\mathsf{in}\, M]\!] = [\![\Gamma, x : \mathsf{var}D \vdash M]\!] \circ \mathsf{cell}_v^x \frown k_{var}$$

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

Language constants and constructs are interpreted as follows:

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

Although it is not important at what position in a word costs are placed, for simplicity we decide to attach them just after the initial move. The only exception is the rule for sequential composition (;), where the cost is placed between two arguments. The reason will be explained later on.

We now show how slot-games model relates to the operational semantics. First, we need to show how to represent the state explicitly in the model. A Γ-state s is interpreted as follows:

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

The regular expression $[\![s]\!]$ is defined over the alphabet $\mathscr{A}^{x_1}_{[\![\mathsf{var}D_1]\!]} + \ldots + \mathscr{A}^{x_k}_{[\![\mathsf{var}D_k]\!]}$, and words in $[\![s]\!]$ are such that projections onto x_i-component are the same as those of suitable initialized $\mathsf{cell}_{\mathsf{s}(x_i)}$ strategies. Note that $[\![s]\!]$ is a regular expression without costs. The interpretation of $\Gamma \vdash M : \mathsf{com}$ at state s is:

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

which is defined over the alphabet $\mathscr{A}_{[\![\mathsf{com}]\!]} + \{Ⓢ\}$. The interpretation $[\![\Gamma \vdash M]\!] \circ [\![s]\!]$ can be studied more closely by considering words in which moves from $\mathscr{A}_{[\![\Gamma]\!]}$ are not hidden. Such words are called *interaction sequences*. For any interaction sequence $run \cdot t \cdot done \bowtie ⓝ$ from $[\![\Gamma \vdash M]\!] \circ [\![\mathsf{s}]\!]$, where t is an even-length word over $\mathscr{A}_{[\![\Gamma]\!]}$, 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 $\mathsf{s}'(x_i)$. For example, let $\mathsf{s} = (x \mapsto 1, y \mapsto 2)$, then the

following interaction: $run \cdot write(5)^y \cdot ok^y \cdot read^x \cdot 1^x \cdot done$ leaves the state $s' = (x \mapsto 1, y \mapsto 5)$. Any two-move word of the form: $run^{x_i} \cdot n^{x_i}$ or $write(n)^{x_i} \cdot ok^{x_i}$ will be referred to as *atomic state operation* of $\mathscr{A}_{[\![\Gamma]\!]}$. The following results are proved in [11] for the full ICA (IA plus parallel composition and semaphores), but they also hold for the restricted fragment of it.

Proposition 1. *If* $\Gamma \vdash M : \{\mathsf{com}, \mathsf{expD}\}$ *and* $\Gamma \vdash M, s \longrightarrow^n M', s'$, *then for each interaction sequence* $i \cdot t$ *from* $[\![\Gamma \vdash M']\!] \circ [\![s']\!]$ *(i is an initial move) there exists an interaction* $i \cdot t_a \cdot t \frown ⓝ \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ *such that* t_a *is an empty word or an atomic state operation of* $\mathscr{A}_{[\![\Gamma]\!]}$ *which leaves the state* s'.

Proposition 2. *If* $\Gamma \vdash M, \mathrm{s} \rightsquigarrow^n M', \mathrm{s}'$ *then* $[\![\Gamma \vdash M']\!] \circ [\![s']\!] \bowtie ⓝ \subseteq [\![\Gamma \vdash M]\!] \circ [\![s]\!]$.

Theorem 1 (Consistency)**.** *If* $M, \mathrm{s} \Downarrow^n$ *then* $\exists w \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ *such that* $\mid w \mid = n$ *and* $w^\dagger = run \cdot done$.

Theorem 2 (Computational Adequacy)**.** *If* $\exists w \in [\![\Gamma \vdash M]\!] \circ [\![s]\!]$ *such that* $\mid w \mid = n$ *and* $w^\dagger = run \cdot done$, *then* $M, \mathrm{s} \Downarrow^n$.

We say that a regular expression R is improved by S, denoted as $R \gtrsim S$, if $\forall w \in R, \exists t \in S$, such that $w^\dagger = t^\dagger$ and $\mid w \mid \geq \mid t \mid$.

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

This shows that the two theories of improvement based on operational and game semantics are identical.

4 Detecting Timing Leaks

In this section slot-game semantics is used to detect whether a term with a secret global variable h can leak information about the initial value of h through its timing behaviour.

For this purpose, we define a special command $\mathsf{skip}^{\#}$ which similarly as skip does nothing, but its slot-game semantics is: $[\![\mathsf{skip}^{\#}]\!] = \{run \cdot \# \cdot done\}$, where # is a new special action, called *delimiter*. Since we verify security of a term by running two copies of the same term one after the other, we will use the command $\mathsf{skip}^{\#}$ to specify the boundary between these two copies. In this way, we will be able to calculate running times of the two terms separately.

Theorem 4. *Let* $h : \mathsf{varD} \mid - \vdash M : \mathsf{com}$ *be a semi-closed term, and* [3]

$$R = [\![k : \mathsf{exp}D \vdash \mathsf{new}_D\, h := k \,\mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h' := k \,\mathsf{in}\, M' : \mathsf{com}]\!] \tag{3}$$

Any word of R *is of the form* $w = w_1 \cdot \# \cdot w_2$ *such that* $\mid w_1 \mid = \mid w_2 \mid$ *iff* M *has no timing leaks, i.e. the fact (2) holds.*

Proof. Suppose that any word $w \in R$ is of the form $w = w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$. Let us analyse the regular expression R defined in (3). We have:

$$\begin{aligned} R = \{ & run \cdot k_{var} \cdot q^k \cdot v^k \cdot w_1 \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot v'^k \cdot w_2 \cdot done \mid \\ & run \cdot w_1 \cdot done \in [\![h \vdash M]\!] \circ \mathsf{cell}^h_v, run \cdot w_2 \cdot done \in [\![h' \vdash M']\!] \circ \mathsf{cell}^{h'}_{v'} \} \end{aligned}$$

for arbitrary values $v, v' \in D$. In order to ensure that one k_{seq} unit of cost occurs before and after the delimiter action, k_{seq} is played between two arguments of the sequential composition as was described in Section 3. Given that $run \cdot w_1 \cdot done \in [\![h \vdash M]\!] \circ \mathsf{cell}^h_v$ and $run \cdot w_2 \cdot done \in [\![h' \vdash M']\!] \circ \mathsf{cell}^{h'}_{v'}$ for any

[3] The free identifier k in (3) is used to initialize the variables h and h' to arbitrary values from D.

$v, v' \in D$, by Computational Adequacy we have that $M, (h \mapsto v) \Downarrow^{|w_1|}$ and $M', (h' \mapsto v') \Downarrow^{|w_2|}$. Since $| w_1 | = | w_2 |$, it follows that the fact (2) holds.

Let us consider the opposite direction. Suppose that the fact (2) holds. The term in (3) is α-equivalent to $k \vdash \mathsf{new}_D\, h := k \,\mathsf{in}\, \mathsf{new}_D\, h' := k \,\mathsf{in}\, M;\ \mathsf{skip}^{\#};\ M'$. Consider $[\![h, h' \vdash M;\ \mathsf{skip}^{\#};\ M']\!] \circ [\![(h \mapsto v) \otimes (h' \mapsto v')]\!]$, where $v, v' \in D$. By Consistency, we have that $\exists w_1 \in [\![h, h' \vdash M]\!] \circ [\![(h \mapsto v) \otimes (h' \mapsto v')]\!]$ such that $| w_1 | = n$ and w_1 leaves the state $(h \mapsto v_1) \otimes (h' \mapsto v')$, and $\exists w_2 \in [\![h, h' \vdash M']\!] \circ [\![(h \mapsto v_1) \otimes (h' \mapsto v')]\!]$ such that $| w_2 | = n$ and w_2 leaves the state $(h \mapsto v_1) \otimes (h' \mapsto v'_1)$. Any word $w \in R$ is obtained from w_1 and w_2 as above ($| w_1 | = | w_2 |$), and so satisfies the requirements of the theorem. □

We can detect timing leaks from a semi-closed term by verifying that all words in the model in (3) are in the required form. To do this, we restrict our attention only to the costs of words in R.

Example 1. Consider the term:

$$h : \mathsf{var}\,\mathsf{int}_2 \vdash \mathsf{if}\,(!h > 0)\,\mathsf{then}\,h := !h + 1;\ \mathsf{else}\,\mathsf{skip} : \mathsf{com}$$

The slot-game semantics of this term extended as in (3) is:

$$\begin{aligned} run \cdot k_{var} \cdot q^k \cdot \big(&0^k \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot (0^k \cdot done + 1^k \cdot k_{der} \cdot k_+ \cdot k_{asg} \cdot done) \\ &+ 1^k \cdot k_{seq} \cdot k_{der} \cdot k_+ \cdot k_{asg} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot (0^k \cdot done + 1^k \cdot k_{der} \cdot k_+ \cdot k_{asg} \cdot done)\big) \end{aligned}$$

This model includes all possible observable interactions of the term with its environment, which contains only the identifier k, along with the costs measuring its running time. Note that the first value for k read from the environment is used to initialize h, while the second value for k is used to initialize h'.

By inspecting we can see that the model contains the word:

$$run \cdot k_{var} \cdot q^k \cdot 0^k \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot 1^k \cdot k_{der} \cdot k_+ \cdot k_{asg} \cdot done$$

which is not of the required form. This word (play) corresponds to two computations of the given term where initial values of h are 0 and 1 respectively, such that the cost of the second computation has additional $k_{der} + k_+ + k_{asg}$ units more than the first one. □

We now show how to detect timing leaks of a split (open) term $h : \mathsf{varD} \mid \Delta \vdash M : \mathsf{com}$, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$. To do this, we need to check timing efficiency of the following model:

$$[\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k];\ \mathsf{skip}^{\#};\ M'[N_1/x_1, \ldots, N_k/x_k]]\!] \tag{4}$$

at state $(h \mapsto v, h' \mapsto v')$, for any closed terms $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$, and for any values $v, v' \in D$. As we have shown slot-game semantics respects theory of operational improvement, so we will need to examine whether all its complete plays-with-costs s are of the form $s_1 \cdot \# \cdot s_2$ where $| s_1 | = | s_2 |$. However, the model in (4) can not be represented as a regular language, so it can not be used directly for detecting timing leaks.

Let us consider more closely the slot-game model in (4). Terms M and M' are run in the same context Δ, which means that each occurrence of a free identifier x_i from Δ behaves uniformly in both M and M'. So any complete play-with-costs of the model in (4) will be a concatenation of complete plays-with-costs from models for M and M' with additional constraints that behaviours of free identifiers from Δ are the same in M and M'. If these additional constraints are removed from the above model, then we generate a model which is an over-approximation of it and where free identifiers from Δ can behave freely in M and M'. Thus we obtain:

$$\begin{aligned} [\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k]; \mathsf{skip}^{\#}; M'[N_1/x_1, \ldots, N_k/x_k]]\!] \subseteq \\ [\![h, h' : \mathsf{varD} \vdash M; \mathsf{skip}^{\#}; M'[N_1/x_1, \ldots, N_k/x_k]]\!] \end{aligned}$$

If $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$ are arbitrary closed terms, then they are interpreted by identity (copy-cat) strategies corresponding to their types, and so we have:

$$[\![h, h' : \mathsf{varD} \vdash M; \mathsf{skip}^{\#}; M'[N_1/x_1, \ldots, N_k/x_k]]\!] = [\![h, h' : \mathsf{varD}, \Delta \vdash M; \mathsf{skip}^{\#}; M']\!]$$

This model is a regular language and we can use it to detect timing leaks.

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

$$S = [\![k : \mathsf{exp}D, \Delta \vdash \mathsf{new}_D\, h := k \,\mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h' := k \,\mathsf{in}\, M' : \mathsf{com}]\!] \tag{5}$$

If any word of S is of the form $w = w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$, Then $h : \mathsf{varD} \mid \Delta \vdash M$ has no timing leaks.

Note that the opposite direction in the above result does not hold. That is, if there exists a word from S which is not of the required form then it does not follow that M has timing leaks, since the found word (play) may be spurious introduced due to over-approximation in the model in (5), and so it may be not present in the model in (4).

Example 2. Consider the term:

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

where f is a non-local call-by-name function.

The slot-game model for this term is as follows:

$$run \cdot k_{app} \cdot run^f \cdot (q^{f,1} \cdot k_{der} \cdot read^h \cdot (0^h \cdot 0^{f,1} + 1^h \cdot 1^{f,1}))^* \cdot done^f \cdot done$$

Once f is called, it may evaluate its argument, zero or more times, and then it terminates successfully. Notice that moves tagged with f represent the actions of calling and returning from the function f, while moves tagged with $f,1$ indicate actions of the first argument of f.

If we generate the slot-game model of this term extended as in (5), we obtain a word which is not in the required form:

$$run \cdot k_{var} \cdot q^k \cdot 0^k \cdot k_{app} \cdot run^f \cdot q^{f,1} \cdot k_{der} \cdot 0^{f,1} \cdot done^f \cdot k_{seq} \cdot \# \cdot k_{seq} \cdot k_{var} \cdot q^k \cdot 1^k \cdot k_{app} \cdot run^f \cdot done^f \cdot done$$

This word corresponds to two computations of the term, where the first one calls f which evaluates its argument once, and the second calls f which does not evaluate its argument at all. The first computation will have the cost of k_{der} units more that the second one. However, this is 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. □

To handle this problem, we can generate an under-approximation of the model given in (4) which can be represented as a regular language. Let $h : \mathsf{varD} \mid \Delta \vdash M$ be a term derived without using the contraction rule for any identifier from Δ. Consider the following model:

$$\begin{aligned}[\![h, h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m = [\![h, h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!] \cap \\ (\mathsf{delta}^{x_1}_{T_1,m} \bowtie \ldots \bowtie \mathsf{delta}^{x_k}_{T_k,m} \bowtie (\mathscr{A}_{[\![h,h':\mathsf{varD} \vdash \mathsf{com}]\!]} + \textcircled{\$})^*)\end{aligned} \tag{6}$$

where $m \geq 0$ denotes the number of times that free identifiers of function types may evaluate its arguments at most. The regular expressions $\mathsf{delta}_{T,m}$ are used to repeat zero or once an arbitrary behaviour for terms of type T, and are defined as follows.

$$\begin{aligned}&\mathsf{delta}_{\mathsf{exp}D,0} = q \cdot \textstyle\sum_{n \in D} n \cdot (\varepsilon + q \cdot n) \qquad \mathsf{delta}_{\mathsf{com},0} = run \cdot done \cdot (\varepsilon + run \cdot done) \\ &\mathsf{delta}_{\mathsf{var}D,0} = (read \cdot \textstyle\sum_{n \in D} n \cdot (\varepsilon + read \cdot n)) \ + \ (\textstyle\sum_{n \in D} write(n) \cdot ok \cdot (\varepsilon + write(n) \cdot ok))\end{aligned}$$

If T is a first-order function type, then $\mathsf{delta}_{T,m}$ will be a regular language only when the number of times its arguments can be evaluated is limited. For example, we have that:

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

If T is a function type with k arguments, then 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.

Notice that we allow an arbitrary behavior of type T to be repeated zero or once in $\mathsf{delta}_{T,m}$, since it is possible that depending on the current value of h an occurrence of a free identifier from Δ to be run in M but not in M', or vice versa. For example, consider the term:

$$h : \mathsf{var}\,\mathsf{int}_2 \mid x, y : \mathsf{exp}\,\mathsf{int}_2 \vdash \mathsf{new}_{int_2}\, z := 0 \,\mathsf{in}\, \mathsf{if}\,(!h > 0)\,\mathsf{then}\, z := x \,\mathsf{else}\, z := y + 1$$

This term has timing leaks, and the corresponding counter-example contains only one interaction with x occurred in a computation, and one interaction with y occurred in the other computation. This counter-example will be included in the model in (6), only if $\mathsf{delta}_{T,m}$ is defined as above.

Let $h : \mathsf{varD} \mid \Delta \vdash M$ be an arbitrary term where identifiers from Δ may occur more than once in M. Let $h : \mathsf{varD} \mid \Delta_1 \vdash M_1$ be derived without using the contraction for Δ_1, such that $h : \mathsf{varD} \mid \Delta \vdash M$ is obtained from it by applying one or more times the contraction rule for identifiers from Δ. Then $[\![h, h' : \mathsf{varD} \mid \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m$ is obtained by first computing $[\![h, h' : \mathsf{varD} \mid \Delta_1 \vdash M_1;\ \mathsf{skip}^{\#};\ M'_1]\!]^m$ as defined in (6), and then by suitable tagging all moves associated with several occurrences of the same identifier from Δ as described in the interpretation of contraction. We have that:

$$[\![h, h' : \mathsf{varD}, \Delta \vdash M;\ \mathsf{skip}^{\#};\ M']\!]^m \subseteq [\![h, h' : \mathsf{varD} \vdash M[N_1/x_1, \ldots, N_k/x_k];\ \mathsf{skip}^{\#};\ M'[N_1/x_1, \ldots, N_k/x_k]]\!]$$

for any $m \geq 0$ and arbitrary closed terms $\vdash N_1 : T_1, \ldots, \vdash N_k : T_k$.

In the case that Δ contains only identifiers of base types B which do not occur in any while-subterm of M, then in the above formula the subset relation becomes the equality for $m = 0$. If a free identifier occurs in a while-subterm of M, then it can be called arbitrary many times in M, and so we cannot reproduce its behaviour in M'.

Theorem 6. *Let $h : \mathsf{varD} \mid \Delta \vdash M$ be a split (open) term, where $\Delta = x_1 : T_1, \ldots, x_k : T_k$, and*

$$T = \ [\![k : \mathsf{exp}D, \Delta \vdash \mathsf{new}_D\, h := k \,\mathsf{in}\, M;\ \mathsf{skip}^{\#};\ \mathsf{new}_D\, h' := k \,\mathsf{in}\, M' : \mathsf{com}]\!]^m \tag{7}$$

(i) Let Δ contains only identifiers of base types B, which do not occur in any while*-subterm of M. Any word of T (where $m = 0$) is of the form $w_1 \cdot \# \cdot w_2$ such that $\mid w_1 \mid = \mid w_2 \mid$ iff M has no timing leaks.*

(ii) Let Δ be an arbitrary context. If there exists a word $w = w_1 \cdot \# \cdot w_2 \in T$ such that $\mid w_1 \mid \neq \mid w_2 \mid$, Then M does have timing leaks.

Note that if a counter-example witnessing a timing leakage is found, then it provides a specific context Δ, i.e. a concrete definition of identifiers from Δ, for which the given open term have timing leaks.

5 Detecting Timing-Aware Non-interference

The slot-game semantics model contains enough information to check the non-interference property of terms along with timing leaks. The method for verifying the non-interference property is analogous to

the one described in [8], where we use the standard game semantics model. As slot-game semantics can be considered as the standard game semantics augmented with the information about quantitative assessment of time usage, we can use it as underlying model for detection of both non-interference property and timing leaks, which we call *timing-aware non-interference*.

In what follows, we show how to verify timing-aware non-interference property for closed terms. In the case of open terms, the method can be extended straightforwardly by following the same ideas for handling open terms described in Section 4.

Let $l : \mathsf{var}D, h : \mathsf{var}D' \vdash M : \mathsf{com}$ be a term where l and h represent low- and high-security global variables respectively. We define $\Gamma_1 = l : \mathsf{var}D, h : \mathsf{var}D'$, $\Gamma_1' = l' : \mathsf{var}D, h' : \mathsf{var}D'$, and M' is α-equivalent to $M[l'/l, h'/h]$ where all bound variables are suitable renamed. We say that $\Gamma_1 \mid - \vdash M : \mathsf{com}$ satisfies *timing-aware non-interference* if

$$\begin{array}{ll} \forall s_1 \in St(\Gamma_1), s_2 \in St(\Gamma_1'). & s_1(l) = s_2(l') \wedge s_1(h) \neq s_2(h') \wedge \\ & \Gamma_1 \vdash M;\ M', \mathrm{s}_1 \otimes \mathrm{s}_2 \rightsquigarrow^{n_1} \mathsf{skip};\ M', \mathrm{s}_1{}' \otimes \mathrm{s}_2 \rightsquigarrow^{n_2} \mathsf{skip};\ \mathsf{skip}, \mathrm{s}_1{}' \otimes \mathrm{s}_2{}' \\ & \Rightarrow s_1'(l) = s_2'(l') \ \wedge \ n_1 = n_2 \end{array}$$

Suppose that abort is a special free identifier of type com^{abort} in Γ. We say that a term $\Gamma \vdash M$ is *safe* iff $\Gamma \vdash M[\mathsf{skip}/\mathsf{abort}] \sqsubseteq_{\approx} M[\mathsf{diverge}/\mathsf{abort}]$ [4]; otherwise we say that a term is *unsafe*. It has been shown in [5] that a term $\Gamma \vdash M$ is safe iff $[\![\Gamma \vdash M]\!]$ does not contain any play with moves from $\mathscr{A}^{abort}_{[\![\mathsf{com}]\!]}$, which we call unsafe plays. For example, $[\![\mathsf{abort} : \mathsf{com}^{\mathsf{abort}} \vdash \mathsf{skip};\ \mathsf{abort} : \mathsf{com}]\!] = run \cdot run^{abort} \cdot done^{abort} \cdot done$, so this term is unsafe.

By using Theorem 4 from Section 4 and the corresponding result for closed terms from [8], it is easy to show the following result.

$$\begin{array}{rl} L = [\![k : \mathsf{exp}D, k' : \mathsf{exp}D', \mathsf{abort} : \mathsf{com} \vdash & \mathsf{new}_D\, l := k \,\mathsf{in}\, \mathsf{new}_{D'}\, h := k' \,\mathsf{in} \\ & \mathsf{new}_D\, l' := !l \,\mathsf{in}\, \mathsf{new}_{D'}\, h' := k' \,\mathsf{in} \\ & \mathsf{skip}^{\#};\ M;\ \mathsf{skip}^{\#};\ M';\ \mathsf{skip}^{\#};\ \mathsf{if}\,(!l \neq !l')\,\mathsf{then}\,\mathsf{abort} : \mathsf{com}]\!] \end{array} \quad (8)$$

The regular expression L contains no unsafe word (plays) and all its words are of the form $w = w_1 \cdot \# \cdot w_2 \cdot \# \cdot w_3 \cdot \# \cdot w_4$ such that $\mid w_2 \mid = \mid w_3 \mid$ iff M satisfies the timing-aware non-interference property.

Notice that the free identifier k in (8) is used to initialize the variables l and l' to any value from D which is the same for both l and l', while k' is used to initialize h and h' to any values from D'. The last if command is used to check values of l and l' in the final state after evaluating the term in (8). If their values are different, then abort is run.

6 Application

We can also represent slot-game semantics model of IA_2 by using the CSP process algebra. This can be done by extending the CSP representation of standard game semantics given in [6], by attaching the costs corresponding to each translation rule. In the same way, we have adapted the verification tool in [6] to automatically convert an IA_2 term into a CSP process [17] that represents its slot-game semantics. The CSP process outputted by our tool is defined by a script in machine readable CSP which can be analyzed by the FDR tool. It represents a model checker for the CSP process algebra, and in this way a range of properties of terms can be verified by calls to it.

[4] $\sqsubseteq_{\approx}$ denotes observational approximation of terms (see [1])

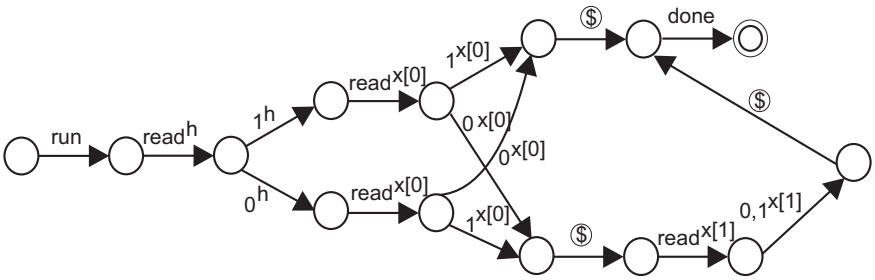

Figure 1: Slot-game semantics for the linear search with k=2

In the input syntax of terms, 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 $max\{n_1,n_2\}$.

In order to use this tool to check for timing leaks in terms, we need to encode the required property as a CSP process (i.e. regular-language). This can be done only if we know the cost of the worst plays (paths) in the model of a given term. We can calculate the worst-case cost of a term by generating its model, and then by counting the number of tokens in its plays. The property we want to check will be: $\sum_{i=0}^{n} ⓘ \cdot \# \cdot ⓘ$, where n denotes the worst-case cost of a term.

To demonstrate practicality of this approach for automated verification, we consider the following implementation of the linear-search algorithm.

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

The meta variable $k > 0$ represents the array size. The term 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 input array. At the moment when the value is found in the array, the term terminates successfully. Note that arrays are introduced in the model as syntactic sugar by using existing term formers. So an array $x[k]$ is represented as a set of k distinct variables $x[0],\ldots,x[k-1]$ (see [6, 10] for details).

Suppose that we are only interested in measuring the efficiency of the term relative to the number of *compare* operations. It is defined as follows $compare : \mathsf{expint}_2 \to \mathsf{expint}_2 \to \mathsf{expbool}$, and its semantics compares for equality the values of two arguments with cost Ⓢ:

$$[\![compare : \mathsf{expint}_2^1 \to \mathsf{expint}_2^2 \to \mathsf{expbool}]\!] = q \cdot Ⓢ \cdot q^1 \cdot (\textstyle\sum_{m \neq n} m^1 \cdot q^2 \cdot n^2 \cdot \mathit{ff}) + (\sum_{m=n} m^1 \cdot q^2 \cdot n^2 \cdot tt)$$

where $m,n \in \{0,1\}$. We assume that the costs of all other operations are relatively negligible (e.g. $k_{var} = k_{der} = \ldots = 0$).

We show the model for this term with $k = 2$ in Fig. 1. The worst-case cost of this term is equal to the array's size k, which occurs when the search fails or the value of h is compared with all elements of the array. We can perform a security analysis for this term by considering the model extended as in (7), where $m = 0$. We obtain that this term has timing leaks, with a counter-example corresponding to two

computations, such that initial values of h are different, and the search succeeds in the one after only one iteration of while and fails in the other. For example, this will happen when all values in the array x are 0's, and the value of h is 0 in the first computation and 1 in the second one.

We can also automatically analyse in an analogous way terms where the array size k is much larger. Also the set of data that can be stored into the global variable h and array x can be larger than $\{0,1\}$. In these cases we will obtain models with much bigger number of states, but they still can be automatically analysed by calls to the FDR tool.

7 Conclusion

In this paper we have described how game semantics can be used for verifying security properties of open sequential programs, such as timing leaks and non-interference. This approach can be extended to terms with infinite data types, such as integers, by using some of the existing methods and tools based on game semantics for verifying such terms. Counter-example guided abstraction refinement procedure (ARP) [5] and symbolic representation of game semantics model [7] are two methods which can be used for this aim. The technical apparatus introduced here applies not only to time as a resource but to any other observable resource, such as power or heating of the processor. They can all be modeled in the framework of slot games and checked for information leaks.

We have focussed here on analysing the IA language, but we can easily extend this approach to any other language for which game semantics exists. Since fully abstract game semantics was also defined for probabilistic [4], concurrent [12], and programs with exceptions [1], it will be interesting to extend this approach to such programs.

References

[1] Abramsky, S., and McCusker, G: Game Semantics. In Proceedings of *the 1997 Marktoberdorf Summer School: Computational Logic* , (1998), 1–56. Springer.

[2] Agat, J: Transforming out Timing Leaks. In: Wegman, M.N., Reps, T.W. (eds.) POPL 2000. ACM, pp. 40–53. ACM, New York (2000), doi:10.1145/325694.325702.

[3] 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), doi:10.1109/CSFW.2004.17.

[4] V. Danos and R. Harmer. Probabilistic Game Semantics. In Proceedings of LICS 2000. 204–213. IEEE Computer Society Press, Los Alamitos (2000), doi:10.1109/LICS.2000.855770.

[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), doi:10.1007/11547662_9.

[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), doi:10.1007/s10009-006-0005-y.

[7] Dimovski, A: Symbolic Representation of Algorithmic Game Semantics. In: Faella, M., Murano, A. (eds.) GandALF 2012. EPTCS vol. 96, pp. 99–112. Open Publishing Association, (2012), doi:10.4204/EPTCS.96.8.

[8] Dimovski, A: Ensuring Secure Non-interference of Programs by Game Semantics. Submitted for publication.

[9] Cartwright, R., Curien, P. L., and Felleisen, M: Fully abstract semantics for observably sequential languages. In *Information and Computation* **111(2)**, pp. 297–401, (1994), doi:10.1006/inco.1994.1047.

[10] Ghica, D. R., McCusker, G: The Regular-Language Semantics of Second-order Idealized Algol. Theoretical Computer Science **309** (1–3), pp. 469–502, (2003), doi:10.1016/S0304-3975(03)00315-3.

[11] 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), doi:10.1145/1040305.1040313.

[12] Ghica, D. R., Murawski, A: Compositional Model Extraction for Higher-Order Concurrent Programs. In: Hermanns, H., Palsberg, J. (eds.) TACAS 2006. LNCS vol. 3920, pp. 303–317. Springer, Heidelberg (2006), doi:10.1007/11691372_20.

[13] Goguen, J., Meseguer, J: Security polices and security models. In: IEEE Symp. on Security and Privacy 1982. pp. 11–20. IEEE Computer Society Press, (1982).

[14] 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), doi:10.1145/268946.268976.

[15] Joshi, R., and Leino, K.R.M: A semantic approach to secure information flow. In *Science of Computer Programming* **37**, pp. 113–138, (2000), doi:10.1016/S0167-6423(99)00024-6.

[16] Reynolds, J. C: The essence of Algol. In: O'Hearn, P.W, and Tennent, R.D. (eds), *Algol-like languages*. (Birkhaüser, 1997).

[17] Roscoe, W. A: *Theory and Practice of Concurrency*. Prentice-Hall, 1998.

[18] Sabelfeld, A., and Myers, A.C: Language-based information-flow security. In IEEE *Journal on Selected Areas in Communications* **21(1)**, (2003), 5–19, doi:10.1109/JSAC.2002.806121.

[19] Sands, D: *Improvement Theory and its Applications*. Cambridge University Press, 1998.

[20] 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, doi:10.3233/JCS-1996-42-304.

[21] 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), doi:10.1109/CSFW.1997.596807.