Available online at www.sciencedirect.com

ScienceDirect

Electronic Notes in Theoretical Computer Science 298 (2013) 29–50

Electronic Notes in Theoretical Computer Science

www.elsevier.com/locate/entcs

Layout Randomization and Nondeterminism

Martín Abadi

Microsoft Research, Silicon Valley and UC Santa Cruz

Jérémy Planul

Stanford University

Gordon Plotkin

LFCS, Informatics, University of Edinburgh and Microsoft Research, Silicon Valley

Abstract

In security, layout randomization is a popular, effective attack mitigation technique. Recent work has aimed to explain it rigorously, focusing on deterministic systems. In this paper, we study layout randomization in the presence of nondeterministic choice. We develop a semantic approach based on denotational models and simulation relations. This approach abstracts from language details, and helps manage the delicate interaction between probabilities and nondeterminism.

Keywords: security, semantics, probabilities, nondeterminism, full abstraction.

1 Introduction

Randomization has important applications in security, ranging from probabilistic cryptographic schemes [10] to the introduction of artificial diversity in low-level software protection [8]. Developing rigorous models and analyses of the systems that employ randomization can be challenging, not only because of the intrinsic difficulty of reasoning about probabilities but also because these systems typically exhibit many other interesting features. Some of these features, such as assumed bounds on the capabilities and the computational complexity of attackers, stem directly from security considerations. Others, such as nondeterminism, need not be specifically related to security, but arise because of the generality of the ambient computational models, which may for example include nondeterministic scheduling for concurrent programs and for network protocols.

http://dx.doi.org/10.1016/j.entcs.2013.09.006

The form of randomization that we explore in this paper is layout randomization in software systems (e.g., [6,18,7]). Layout randomization refers to a body of widely used techniques that place data and code randomly in memory. In practice, these techniques effectively thwart many attacks that assume knowledge of the location of data and code. Recent research by the authors and others aims to develop rigorous models and proofs for layout randomization [19,3,13,2]. The research to date has focused on deterministic, sequential programs. Here, we consider layout randomization for programs that may make nondeterministic choices.

We phrase our study in terms of a high-level language in which variables are abstract (symbolic) locations, and a low-level language in which they are mapped to random natural-number addresses in memory. Both languages include a standard construct for nondeterministic choice. We give models for the languages. For each language, we also define a contextual implementation relation. Intuitively, a context may represent an attacker, so contextual implementation relations may serve, in particular, for expressing standard security properties. We characterize contextual implementation relations in terms of semantic simulation relations (so-called logical relations). Throughout, the low-level relations are probabilistic. Via the simulation relations, we obtain a semantic correspondence between the high-level and low-level worlds. Basically, simulation relations in one world induce simulation relations in the other, and therefore contextual implementation in one world implies contextual implementation in the other.

Thus, our approach emphasizes semantic constructions. In comparison with prior syntactic work, arguments via models arguably lead to more satisfying security arguments, independent of superficial details of particular languages (as layout randomization is largely language-agnostic in practice). They also help reconcile probabilities and nondeterminism, which have a rich but thorny interaction.

Some of the difficulties of this interaction have been noticed in the past. For instance, in their development of a framework for the analysis of security protocols [15, Section 2.7], Lincoln et al. observed:

> our intention is to design a language of communicating processes so that an adversary expressed by a set of processes is restricted to probabilistic polynomial time. However, if we interpret parallel composition in the standard nondeterministic fashion, then a pair of processes may nondeterministically "guess" any secret information.

They concluded:

> Therefore, although nondeterminism is a useful modeling assumption in studying correctness of concurrent programs, it does not seem helpful for analyzing cryptographic protocols.

Thus, they adopted a form of probabilistic scheduling, and excluded nondeterminism. In further work, Mitchell et al. [17] refined the framework, in particular defining protocol executions by reference to any polynomial-time probabilistic scheduler that operates uniformly over certain kinds of choices. The uniformity prevents collusion between the scheduler and an attacker. Similarly, Canetti et al. [4] resolved

nondeterminism by task schedulers, which do not depend on dynamic information generated during probabilistic executions; they thus generated sets of trace distributions, one for each task schedule.

From a semantic perspective, a nondeterministic program denotes a function that produces a set of possible outcomes; equally, a probabilistic program represents a function that produces a distribution over outcomes. Rigorous versions of these statements can be cast in terms of powerdomains and probabilistic powerdomains [9]. In principle, a nondeterministic and probabilistic program may represent either a function producing a set of distributions over outcomes or else one producing a distribution over sets of outcomes. However it seems that only the former option, where nondeterministic choice is resolved before probabilistic choice, leads to a satisfactory theory if, for example, one wishes to retain all the usual laws for both forms of nondeterminism [16,21,11].

To illustrate these options, imagine a two-player game in which Player I chooses a bit b_{I} at random, Player II chooses a bit b_{II} nondeterministically, and Player I wins if and only if $b_{\mathrm{I}} = b_{\mathrm{II}}$. The system composed of the two players may be seen as producing a set of distributions or a distribution on sets of outcomes.

- With the former view, we can say that, in each possible distribution, Player I wins with probability 1/2.
- On the other hand, with the latter view, we can say only that, with probability 1, Player I may win and may lose.

The former view is preferable in a variety of security applications, in which we may wish to say that no matter what an attacker does, or how nondeterministic choices are resolved, some expected property holds with high probability.

However, in our work, it does not suffice to resolve nondeterministic choice before probabilistic choice, as we explain in detail below, fundamentally because the probabilistic choices that we treat need not be independent. Instead, we construct a more sophisticated model that employs random variables, here maps from memory layouts to outcomes. The memory layouts form the sample space of the random variables, and, as usual, one works relative to a given distribution over the sample space.

Beyond the study of layout randomization, it seems plausible that an approach analogous to ours could be helpful elsewhere in security analysis. Our models may also be of interest on general grounds, as a contribution to a long line of research on programming-language semantics for languages with nondeterministic and probabilistic choice. Specifically, the models support a treatment of dependent probabilistic choice combined with nondeterminism, which as far as we know has not been addressed in the literature. Finally, the treatment of contextual implementation relations and simulation relations belongs in a long line of research on refinement.

Contents

In Section 2 we review some preliminary material on cpos.

In Section 3, we consider a high-level language, with abstract locations, stan-

dard imperative constructs, and nondeterminism, and describe its denotational and operational semantics. We define a contextual implementation relation with respect to contexts that represent attackers, which we call public contexts; for this purpose, we distinguish public locations, which attackers can access directly, from private locations. We also define a simulation relation, and prove that it coincides with the contextual implementation relation. The main appeal of the simulation relation, as usual, is that it does not require reasoning about all possible contexts.

In Section 4, we similarly develop a lower-level language in which programs may use natural-number memory addresses (rather than abstract locations). Again, we define a denotational semantics, an operational semantics, a contextual implementation relation, and a simulation relation. These definitions are considerably more delicate than those of the high-level language, in particular because they refer to layouts, which map abstract locations to concrete natural-number addresses, and which may be chosen randomly (so we often make probabilistic statements).

In Section 5, we relate the high-level and the low-level languages. We define a simple compilation function that maps from the former to the latter. We then establish that if two high-level commands are in the contextual implementation relation, then their low-level counterparts are also in the contextual implementation relation. The proof leverages simulation relations. In semantics parlance, this result is a full-abstraction theorem; the use of public contexts that represent attackers, however, is motivated by security considerations, and enable us to interpret this theorem as providing a formal security guarantee for the compilation function, modulo a suitable random choice of memory layouts.

Finally, in Section 6 we conclude by discussing some related and further work.

2 Preliminaries on cpos

We take a cpo to be a partial order P closed under increasing ω-sups, and consider sets to be cpos with the discrete ordering. We write $P_\perp$ for the lift of P, viz. P extended by the addition of a least element, $\perp$. Products $P \times Q$ and function spaces $P \to Q$ (which we may also write as Q^P) are defined as usual, with the function space consisting of all continuous functions (those monotonic functions preserving the ω-lubs).

We use the lower, or Hoare, powerdomain $\mathcal{H}(P)$ of the nonempty, downwards, and ω-sup-closed subsets of P, ordered by inclusion. The lower powerdomain is the simplest of the three powerdomains, and models "may" or "angelic" nondeterminism; the others (upper and convex) may also be worth investigating.

For any nonempty subset X of P, we write $X\downarrow$ for the downwards closure $\{y \mid \exists x \in X.\, y \leq x\}$ of X. We also write X^* for the downwards and ω-sup closure of X (which is typically the same as $X\downarrow$ in the instances that arise below).

Both $\mathcal{H}(-)$ and $\mathcal{H}(-_\perp)$ are monads (those for lower nondeterminism, and lower nondeterminism and nontermination, respectively). The unit of the former is $x \mapsto \{x\}\downarrow$ and any continuous map $f : P \to \mathcal{H}(Q)$ has an extension $f^\dagger : \mathcal{H}(P) \to \mathcal{H}(Q)$

given by:

$$f^\dagger(X) = (\bigcup_{x \in X} f(x))^*$$

For the latter the unit is $x \mapsto \{x\}\downarrow$ and the extension $f^\dagger : \mathcal{H}(P_\perp) \to \mathcal{H}(Q_\perp)$ of a continuous map $f : P \to \mathcal{H}(Q_\perp)$ is given by:

$$f^\dagger(X) = \{\perp\} \cup (\bigcup_{x \in X \setminus \{\perp\}} f(x))^*$$

3 The high-level language

In this section, we define our high-level language. In this language, locations are symbolic names, and we use an abstract store to link those locations to their contents, which are natural numbers.

For simplicity, the language lacks data structures and higher-order features. Therefore, locations cannot contain arrays or functions (cf. [2]), except perhaps through encodings. So the language does not provide a direct model of overflows and code-injection attacks, for instance.

There are many other respects in which our languages and their semantics are not maximally expressive, realistic, and complex. They are however convenient for our study of nondeterminism and of the semantic approach to layout randomization.

3.1 Syntax and informal semantics

The syntax of the high-level language includes categories for natural-number expressions, boolean expressions, and commands:

$$\begin{aligned} e &::= k \mid !l_{\mathtt{loc}} \mid e + e \mid e * e \\ b &::= e \leq e \mid \neg b \mid \mathrm{tt} \mid \mathrm{ff} \mid b \vee b \mid b \wedge b \\ c &::= l_{\mathtt{loc}} := e \mid \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c \mid \mathtt{skip} \mid c; c \mid c + c \mid \mathtt{while}\, b\, \mathtt{do}\, c \end{aligned}$$

where k ranges over numerals, and l over a given finite set of store locations Loc. Natural-number expressions are numerals, dereferencing of memory locations, sums, or products. Boolean expressions are inequalities on natural-number expressions, negations, booleans, disjunctions, or conjunctions. Commands are assignments at a location, conditionals, skip, sequences, nondeterministic choices, or loops. Command contexts $C[\ ]$ are commands with holes; we write $C[c]$ for the command obtained by filling all the holes in $C[\ ]$ with c. We further use trivial extensions of this language, in particular with additional boolean and arithmetic expressions.

We assume that the set of store locations Loc is the union of two disjoint sets of locations PubLoc (public locations) and PriLoc (private locations). Let c be a command or a command context. We say that c is public if it does not contain any occurrence of $l_{\mathtt{loc}} := v$ or $!l_{\mathtt{loc}}$ for $l \in \mathrm{PriLoc}$. As in previous work [3], we model

$$[\![l_{\mathtt{loc}} := e]\!](s) = \eta(s[l \mapsto [\![e]\!](s)]) \qquad [\![\mathsf{skip}]\!](s) = \eta(s)$$

$$[\![\mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c']\!](s) = \begin{cases} [\![c]\!](s) & [\![b]\!](s) = \mathrm{tt} \\ [\![c']\!](s) & [\![b]\!](s) = \mathrm{ff} \end{cases} \qquad \begin{aligned} [\![c; c']\!](s) &= [\![c']\!]^\dagger([\![c]\!](s)) \\ [\![c + c']\!](s) &= [\![c]\!](s) \cup [\![c']\!](s) \end{aligned}$$

$$[\![\mathtt{while}\, b\, \mathtt{do}\, c]\!] = \mu\, \theta : S \to \mathcal{H}(S_\perp).\, \lambda s : S. \begin{cases} \eta(s) & ([\![b]\!](s) = \mathrm{ff}) \\ \theta^\dagger([\![c]\!](s)) & ([\![b]\!](s) = \mathrm{tt}) \end{cases}$$

Fig. 1. High-level denotational semantics

attackers by such public commands and command contexts; thus, attackers have direct access to public locations but not, by default, to private locations.

The distinction between public and private locations is directly analogous to that between external and internal state components in automata and other specification formalisms (e.g., [1]). It also resembles distinctions in information-flow systems, which often categorize variables into levels (e.g., [20]), and typically aim to prevent flows of information from "high" to "low" levels. We do not impose any such information-flow constraint: we permit arbitrary patterns of use of public and private locations. Nevertheless, we sometimes use h for a private location and l for a public location, and also associate the symbols H and L with private and public locations, respectively.

3.2 Denotational semantics

A store s is a function from a finite set Loc of store locations to natural numbers. When Loc consists of h and l, for example, we write $(h \mapsto m, l \mapsto n)$ for the store that maps h to m and l to n. A public (private) store is a function from PubLoc (PriLoc) to natural numbers. We write S for the set of stores, S_L for the set of public stores, and S_H for the set of private stores. Note the natural functions:

$$S_L \xleftarrow{L} S \xrightarrow{H} S_H$$

We write s_L for $L(s)$ and $s =_L s'$ when $s_L = s'_L$, and similarly for H.

The denotational semantics

$$[\![e]\!] : \mathrm{Store} \to \mathbb{N} \qquad [\![b]\!] : \mathrm{Store} \to \mathbb{B}$$

of expressions are defined as usual with, in particular, $[\![!l_{\mathtt{loc}}]\!](s) = s(l)$. The denotational semantics

$$[\![c]\!] : S \to \mathcal{H}(S_\perp)$$

of commands is given in Figure 1, where the semantics of the while loop is the standard least-fixed point one.

$$\langle l_{\mathtt{loc}} := e, s\rangle \to s[l \mapsto [\![e]\!]_s] \qquad \frac{[\![b]\!]_s = \mathrm{tt}}{\langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', s\rangle \to \langle c, s\rangle}$$

$$\frac{[\![b]\!]_s = \mathrm{ff}}{\langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', s\rangle \to \langle c', s\rangle} \qquad \langle \mathsf{skip}, s\rangle \to s \qquad \frac{\langle c, s\rangle \to \langle c', s'\rangle}{\langle c; c'', s\rangle \to \langle c'; c'', s'\rangle}$$

$$\frac{\langle c, s\rangle \to s'}{\langle c; c'', s\rangle \to \langle c'', s'\rangle} \qquad \langle c + c', s\rangle \to \langle c, s\rangle \qquad \langle c + c', s\rangle \to \langle c', s\rangle$$

$$\frac{[\![b]\!]_s = \mathrm{ff}}{\langle \mathtt{while}\, b\, \mathtt{do}\, c, s\rangle \to s} \qquad \frac{[\![b]\!]_s = \mathrm{tt}}{\langle \mathtt{while}\, b\, \mathtt{do}\, c, s\rangle \to \langle c; \mathtt{while}\, b\, \mathtt{do}\, c, s\rangle}$$

Fig. 2. High-level operational semantics

Example 3.1 Consider the two commands:

$$c_0 = (h := \mathrm{tt}; l := \neg !l) + (h := \mathrm{ff}) \qquad c_1 = (h := \mathrm{tt}; l := \mathrm{tt}) + (h := \mathrm{ff}; l := \mathrm{ff})$$

According to the semantics, $[\![c_0]\!]$ maps any store mapping l to tt to the set $\{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{ff}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{tt})\}\downarrow$, and any store where l is ff to the set $\{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{tt}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{ff})\}\downarrow$, while $[\![c_1]\!]$ maps any store to the set $\{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{tt}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{ff})\}\downarrow$. In sum, we may write:

$$\begin{aligned}
[\![c_0]\!](h \mapsto _, l \mapsto \mathrm{tt}) &= \{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{ff}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{tt})\}\downarrow \\
[\![c_0]\!](h \mapsto _, l \mapsto \mathrm{ff}) &= \{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{tt}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{ff})\}\downarrow \\
[\![c_1]\!](h \mapsto _, l \mapsto _) &= \{(h \mapsto \mathrm{tt}, l \mapsto \mathrm{tt}), (h \mapsto \mathrm{ff}, l \mapsto \mathrm{ff})\}\downarrow
\end{aligned}$$

Note that the semantics of the two commands are different. Nevertheless, below we show that these two commands are in a sense equivalent (with respect to public contexts). □

3.3 Operational semantics

The high-level language has a straightforward small-step operational semantics. In this semantics, a high-level state is a pair $\langle c, s\rangle$ of a command and a store or, in case of termination, just a store s. The transition relation $\to$ is a binary relation on such states. Figure 2 gives the rules for $\to$.

Proposition 3.2 (Operational/denotational consistency) *Let c be a command and s be a store. We have*

$$[\![c]\!](s) = \{s' | \langle c, s\rangle \to^* s'\} \cup \bot$$

3.4 Implementation relations and equivalences

3.4.1 Contextual pre-order

We introduce a contextual pre-order $\sqsubseteq_L$ on commands. Intuitively, $c \sqsubseteq_L c'$ may be interpreted as saying that c "refines" (or "implements") c', in the sense that the publicly observable outcomes that c can produce are a subset of those that c' permits, in every public context and from every initial store. Thus, let $f = [\![C[c]]\!]$ and $f' = [\![C[c']]\!]$ for an arbitrary public context C, and let s_0 be a store; then for every store s in $f(s_0)$ there is a store s' in $f'(s_0)$ that coincides with s on public locations. Note that we both restrict attention to public contexts and compare s and s' only on public locations.

We define $\sqsubseteq_L$ and some auxiliary relations as follows:

- For $X \in \mathcal{H}(S_\perp)$, we set:

$$X_L = \{s_L \mid s \in X \setminus \perp\} \cup \{\perp\}$$

- For $f, f' : S \to \mathcal{H}(S_\perp)$, we write that $f \leq_L f'$ when, for every store s_0, we have $f(s_0)_L \leq f'(s_0)_L$.
- Let c and c' be two commands. We write that $c \sqsubseteq_L c'$ when, for every public command context C, we have $[\![C[c]]\!] \leq_L [\![C[c']]\!]$.

Straightforwardly, this contextual pre-order relation yields a notion of contextual equivalence with respect to public contexts.

3.4.2 Simulation

In addition to a contextual pre-order, we introduce a simulation relation $\preceq$ whose main advantage, as usual, is that it does not require reasoning about contexts.

As in much previous work, one might expect a simulation relation between two commands c and c' to be a relation on stores that respects the observable parts of these stores, and such that if s_0 is related to s_1 and c can go from s_0 to s'_0 then there exists s'_1 such that s'_0 is related to s'_1 and c' can go from s_1 to s'_1. In our setting, respecting the observable parts of stores means that related stores give the same values to public locations (much like refinement mappings preserve externally visible state components [1], and low-bisimulations require equivalence on low-security variables [20]).

Although this idea could lead to a sound proof technique for the contextual pre-order, it does not suffice for completeness. Indeed, forward simulations, of the kind just described, are typically incomplete on their own for nondeterministic systems. They can be complemented with techniques such as backward simulation, or generalized (e.g., [1,14,5]).

Here we develop one such generalization. Specifically, we use relations on sets of stores. We build them from relations over $\mathcal{H}(S_{H\perp})$ as a way of ensuring the condition that public locations have the same values, mentioned above. We also require other standard closure conditions. Our relations are similar to the ND measures of Klarlund and Schneider [14]. Their work takes place in an automata-

theoretic setting; automata consist of states (which, intuitively, are private) and of transitions between those states, labeled by events (which, intuitively, are public). ND measures are mappings from states to sets of finite sets of states, so can be seen as relations between states and finite sets of states. The finiteness requirement, which we do not need, allows a fine-grained treatment of infinite execution paths via König's Lemma.

First, we extend relations R over $\mathcal{H}(S_{H\perp})$ to relations R^+ over $\mathcal{H}(S_\perp)$, as follows. For any $X \in \mathcal{H}(S_\perp)$ and $s \in S_L$, we define $X_s \in \mathcal{H}(S_{H\perp})$ by:

$$X_s = \{s'_H \mid s' \in X, s'_L = s\} \cup \{\perp\}$$

and then we define R^+ by:

$$XR^+Y \equiv_{\text{def}} \forall s \in S_L.\, (X_s \neq \{\perp\} \Rightarrow Y_s \neq \{\perp\}) \wedge X_s R Y_s$$

If R is reflexive (respectively, is closed under increasing ω-sups; is right-closed under $\leq$; is closed under binary unions) the same holds for R^+. Also, if XR^+Y then $X_L \leq Y_L$.

For any $f, f' : S_\perp \to \mathcal{H}(S_\perp)$ and relation R over $\mathcal{H}(S_{H\perp})$ we write that $f \preceq_R f'$ when:

$$\forall X, Y \in \mathcal{H}(S_\perp).\, XR^+Y \Rightarrow f^\dagger(X) R^+ f'^\dagger(Y)$$

Finally, we write that $f \preceq f'$ if $f \preceq_R f'$ for some reflexive R closed under increasing ω-sups, right-closed under $\leq$, and closed under binary unions.

3.4.3 Contextual pre-order vs. simulation

The contextual pre-order coincides with the simulation relation:

Theorem 3.3 *Let c and c' be two commands of the high-level language. Then $c \sqsubseteq_L c'$ holds if and only if $[\![c]\!] \preceq [\![c']\!]$ does.*

Example 3.4 We can verify that c_0 and c_1, introduced in Example 3.1, are equivalent (with R the full relation). For instance, let $S_0 = \{(h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}\downarrow$ and $S_1 = \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt})\}\downarrow$. We have $S_0 R^+ S_1$, and:

$$[\![c_0]\!]^\dagger(S_0) = \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{ff}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{tt})\}\downarrow$$
$$[\![c_1]\!]^\dagger(S_1) = \{(h \mapsto \mathtt{tt}, l \mapsto \mathtt{tt}), (h \mapsto \mathtt{ff}, l \mapsto \mathtt{ff})\}\downarrow$$

We can then check that:

$$[\![c_0]\!]^\dagger(S_0) R^+ [\![c_1]\!]^\dagger(S_1)$$

□

Example 3.5 In this example, we study the two commands

$$c_2 = \mathtt{if}\, h = 0\, \mathtt{then}\, l := 1\, \mathtt{else}\, (h := 0) + (h := !h - 1)$$
$$c_3 = \mathtt{if}\, h = 0\, \mathtt{then}\, l := 1\, \mathtt{else}\, (h := 0) + \mathsf{skip}$$

which seem to share the same behavior on public variables, but that are inherently different because of their behavior on private variables. According to the semantics, we have:

$$\begin{aligned}
[\![c_2]\!](h \mapsto 0, l \mapsto _) &= \{(h \mapsto 0, l \mapsto 1)\}\downarrow \\
[\![c_2]\!](h \mapsto j+1, l \mapsto k) &= \{(h \mapsto j, l \mapsto k), (h \mapsto 0, l \mapsto k)\}\downarrow \\
[\![c_3]\!](h \mapsto 0, l \mapsto _) &= \{(h \mapsto 0, l \mapsto 1)\}\downarrow \\
[\![c_3]\!](h \mapsto j+1, l \mapsto k) &= \{(h \mapsto j+1, l \mapsto k), (h \mapsto 0, l \mapsto k)\}\downarrow
\end{aligned}$$

We can verify that $c_2 \preceq_R c_3$, with R defined as the smallest relation that satisfies our conditions (reflexivity, etc.) and such that

$$\{(h \mapsto k)\} R \{(h \mapsto k')\} \qquad \text{for all } k \leq k'$$

For instance, suppose $S_0 = \{(h \mapsto 5, l \mapsto 0)\}\downarrow$ and $S_1 = \{(h \mapsto 7, l \mapsto 0)\}\downarrow$. We have $S_0 R^+ S_1$, and:

$$\begin{aligned}
[\![c_2]\!]^\dagger(S_0) &= \{(h \mapsto 4, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow \\
[\![c_3]\!]^\dagger(S_1) &= \{(h \mapsto 7, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow
\end{aligned}$$

We can then check that:

$$[\![c_2]\!]^\dagger(S_0) R^+ [\![c_3]\!]^\dagger(S_1)$$

On the other hand, there is no suitable relation R such that $c_3 \preceq_R c_2$. If there were such a relation R, it would be reflexive, so $\{(h \mapsto 1)\}\ R\ \{(h \mapsto 1)\}$. Suppose that $S_0 = \{(h \mapsto 1, l \mapsto 0)\}\downarrow$ and that $S_1 = \{(h \mapsto 1, l \mapsto 0)\}\downarrow$. We have $S_0 R^+ S_1$, and:

$$\begin{aligned}
[\![c_3]\!]^\dagger(S_0) &= \{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow \\
[\![c_2]\!]^\dagger(S_1) &= \{(h \mapsto 0, l \mapsto 0)\}\downarrow
\end{aligned}$$

We need

$$\{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow\ R^+ \{(h \mapsto 0, l \mapsto 0)\}\downarrow$$

hence $\{(h \mapsto 1)\} R \{(h \mapsto 0)\}$. Now take $S_2 = \{(h \mapsto 1, l \mapsto 0)\}\downarrow$ and $S_3 = \{(h \mapsto 0, l \mapsto 0)\}\downarrow$. We have $S_2 R^+ S_3$, and:

$$\begin{aligned}
[\![c_3]\!]^\dagger(S_2) &= \{(h \mapsto 1, l \mapsto 0), (h \mapsto 0, l \mapsto 0)\}\downarrow \\
[\![c_2]\!]^\dagger(S_3) &= \{(h \mapsto 0, l \mapsto 1)\}\downarrow
\end{aligned}$$

Since the values of l do not match, we cannot have $[\![c_3]\!]^\dagger(S_2) R^+ [\![c_2]\!]^\dagger(S_3)$, hence $c_3 \not\preceq_R c_2$.

As predicted by Theorem 3.3, we also have $c_3 \not\sqsubseteq_L c_2$. Indeed, for $C = \_\,;\_$ and $s_0 = (h \mapsto 1, l \mapsto 0)$, we have $[\![C[c_3]]\!](s_0) \not\leq_L [\![C[c_2]]\!](s_0)$. □

4 The low-level language

In this section, we define our low-level language. In this language, we use concrete natural-number addresses for memory. We still use abstract location names, but those are interpreted as natural numbers (according to a memory layout), and can appear in arithmetic expressions.

4.1 Syntax and informal semantics

The syntax of the low-level language includes categories for natural-number expressions, boolean expressions, and commands:

$$
\begin{array}{rcl}
e & ::= & k \mid l_{\mathtt{nat}} \mid !e \mid e + e \mid e * e \\
b & ::= & e \leq e \mid \neg b \mid \mathtt{tt} \mid \mathtt{ff} \mid b \vee b \mid b \wedge b \\
c & ::= & e := e \mid \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c \mid \mathtt{skip} \mid c; c \mid c + c \mid \mathtt{while}\, b\, \mathtt{do}\, c
\end{array}
$$

where k ranges over numerals, and l over the finite set of store locations. Boolean expressions are as in the high-level language. Natural-number expressions and commands are also as in the high-level language, except for the inclusion of memory locations among the natural-number expressions, and for the dereferencing construct $!e$ and assignment construct $e := e'$ where e is an arbitrary natural-number expression (not necessarily a location).

Importantly, memory addresses are natural numbers, and a memory is a partial function from those addresses to contents. We assume that accessing an address at which the memory is undefined constitutes an error that stops execution immediately. In this respect, our language relies on the "fatal-error model" of Abadi and Plotkin [3]. With more work, it may be viable to treat also the alternative "recoverable-error model", which permits attacks to continue after such accesses, and therefore requires a bound on the number of such accesses.

4.2 Denotational semantics

4.2.1 Low-level memories, layouts, and errors

We assume given a natural number $\mathrm{r} > |\mathrm{Loc}|$ that specifies the size of the memory. A memory m is a partial function from $\{1, \ldots, \mathrm{r}\}$ to natural numbers; we write Mem for the set of memories. A memory layout w is an injection from Loc to $\{1, \ldots, \mathrm{r}\}$. We consider only memory layouts that extend a given public memory layout w_p (an injection from PubLoc to $\{1, \ldots, \mathrm{r}\}$), fixed in the remaining of the paper. We let W be the set of those layouts.

The security of layout randomization depends on the randomization itself. We let d be a probability distribution on memory layouts (that extend w_p). When φ is a predicate on memory layouts, we write $P_d(\varphi(w))$ for the probability that $\varphi(w)$ holds with w sampled according to d.

Given a distribution d on layouts, we write δ_d for the minimum probability for

a memory address to have no antecedent location (much as in [3]):

$$\delta_d = \min_{i \in \{1,\ldots,\mathrm{r}\} \setminus \mathrm{ran}(w_p)} P_d(i \notin \mathrm{ran}(w))$$

We assume that $\delta_d > 0$. This probability bounds 1 minus the maximum probability for an adversary to guess a location. For common distributions (e.g., the uniform distribution), δ_d approaches 1 as r grows, indicating that adversaries fail most of the time. We assume d fixed below, and may omit it, writing δ for δ_d.

The denotational semantics of the low-level language uses the "error + nontermination" monad $P_{\xi\perp} =_{\mathrm{def}} (P + \{\xi\})_\perp$, which first adds an "error" element ξ to P and then a least element. As the monad is strong, functions $f : P_1 \times \ldots \times P_n \to Q_{\xi\perp}$ extend to functions $\overline{f}$ on $(P_1)_{\xi\perp} \times \ldots \times (P_n)_{\xi\perp}$, where $\overline{f}(x_1, \ldots, x_n)$ is ξ or $\perp$ if some x_j, but no previous x_i, is; we write f for $\overline{f}$.

For any memory layout w and store s, we let $w \cdot s$ be the memory defined on $\mathrm{ran}(w)$ by:

$$w \cdot s(i) = s(l) \text{ for } w(l) = i$$

The notation $w \cdot s$ extends to $s \in S_{\xi\perp}$, as above, so that $w \cdot \xi = \xi$ and $w \cdot \perp = \perp$. A store projection is a function $\zeta : \mathrm{Mem}_{\xi\perp}^{W}$ of the form $w \mapsto w \cdot s$, for some $s \in S_{\xi\perp}$.

4.2.2 What should the denotational semantics be?

We discuss a simple example in order to explain our choice of type of the low-level denotational semantics. A straightforward semantics might have the type:

$$W \times \mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\perp})$$

so that the meaning of a command would be a function from layouts and memories to sets of memories (modulo the use of the "error + nontermination" monad). Using our example we argue that this is unsatisfactory, and arrive at a more satisfactory alternative.

Suppose that there is a unique private location l, and that memory has four addresses, $\{1, 2, 3, 4\}$. We write s_i for the store $(l \mapsto i)$. The 4 possible layouts are $w_i = (l \mapsto i)$, for $i = 1, \ldots, 4$. Assume that d is uniform. Consider the following command:

$$c_4 \;=\; (1{:=}1) \;+\; (2{:=}1) \;+\; (3{:=}1) \;+\; (4{:=}1)$$

which nondeterministically guesses an address and attempts to write 1 into it. Intuitively, this command should fail to overwrite l most of the time. However, in a straightforward semantics of the above type we would have:

$$[\![c_4]\!](w_j, w_j \cdot s_0) = \{\xi, w_j \cdot s_1\} \downarrow$$

and we cannot state any quantitative property of the command, only that it sometimes fails and that it sometimes terminates.

One can rewrite the type of this semantics as:

$$\mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\perp})^{W}$$

and view that as a type of functions that yield an $\mathcal{H}(\mathrm{Mem}_{\xi\perp})$-valued random variable with sample space W (the set of memory layouts) and distribution d. Thus, in this semantics, the nondeterministic choice is made after the probabilistic one —the wrong way around, as indicated in the Introduction.

It is therefore natural to reverse matters and look for a semantics of type:

$$\mathrm{Mem} \to \mathcal{H}(\mathrm{Mem}_{\xi\perp}^{W})$$

now yielding a set of $\mathrm{Mem}_{\xi\perp}$-valued random variables—so, making the nondeterministic choice first. Desirable as this may be, there seems to be no good notion of composition of such functions.

Fortunately, this last problem can be overcome by changing the argument type to also be that of $\mathrm{Mem}_{\xi\perp}$-valued random variables:

$$\mathrm{Mem}_{\xi\perp}^{W} \to \mathcal{H}(\mathrm{Mem}_{\xi\perp}^{W})$$

It turns out that with this semantics we have:

$$[\![c_4]\!](\zeta_i) = \{\zeta_\xi^1, \zeta_\xi^2, \zeta_\xi^3, \zeta_\xi^4\}\downarrow$$

where $\zeta_i(w) = w \cdot s_i$ and $\zeta_\xi^i(w) = w_i \cdot s_1$ if $w = w_i$ and $= \xi$ otherwise. We can then say that, for every nondeterministic choice, the probability of an error (or nontermination, as we are using the lower powerdomain) is 0.75.

In a further variant in the definition of the semantics, one might replace $\mathrm{Mem}_{\xi\perp}$-valued random variables by the corresponding probability distributions on $\mathrm{Mem}_{\xi\perp}$, via the natural map $\mathrm{Ind}_d : \mathrm{Mem}_{\xi\perp}^{W} \longrightarrow \mathcal{V}(\mathrm{Mem}_{\xi\perp})$ induced by the distribution d on W. Such a semantics could have the form:

$$\mathrm{Mem} \to \mathcal{H}_{\mathcal{V}}(\mathrm{Mem}_{\xi\perp})$$

mapping memories to probability distributions on memories, where $\mathcal{H}_{\mathcal{V}}$ is a powerdomain for mixed nondeterministic and probabilistic choice as discussed above. However, such an approach would imply (incorrectly) that a new layout is chosen independently for each memory operation, rather than once and for all. In our small example with the single private location l and four addresses, it would not capture that $(1:=1);(2:=1)$ will always fail. It would treat the two assignments in $(1:=1);(2:=1)$ as two separate guesses that may both succeed. Similarly, it would treat the two assignments in $(1 := 1);(1 := 2)$ as two separate guesses where the second guess may fail to overwrite l even if the first one succeeds. With a layout chosen once and for all, on the other hand, the behavior of the second assignment is completely determined after the first assignment.

4.2.3 Denotational semantics

The denotational semantics

$$[\![e]\!] : \mathrm{Mem} \times W \to \mathbb{N}_{\xi\perp} \qquad [\![b]\!] : \mathrm{Mem} \times W \to \mathbb{B}_{\xi\perp}$$

$$[\![c + c']\!](\zeta) = [\![c]\!](\zeta) \cup [\![c']\!](\zeta) \qquad [\![c; c']\!] = [\![c']\!]^\dagger \circ [\![c]\!] \qquad [\![\mathsf{skip}]\!] = \eta$$
$$[\![e := e']\!](\zeta) = \eta(\lambda w : W.\, \mathrm{Ass}(\zeta(w), [\![e]\!]^w_{\zeta(w)}, [\![e']\!]^w_{\zeta(w)}))$$
$$[\![\mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c']\!] = \mathrm{Cond}([\![b]\!], [\![c]\!], [\![c']\!])$$
$$[\![\mathtt{while}\, b\, \mathtt{do}\, c]\!] = \mu\theta : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp}).\mathrm{Cond}([\![b]\!], \theta^\dagger \circ [\![c]\!], \eta)$$

Fig. 3. Low-level denotational semantics

of expressions are defined in a standard way, with, in particular, $[\![l_{\mathtt{nat}}]\!]^w_m = w(l)$, and also $[\![!e]\!]^w_m = m([\![e]\!]^w_m)$, if $[\![e]\!]^w_m \in \mathrm{dom}(m)$, and $= \xi$, otherwise, using an obvious notation for functional application. Note that these semantics never have value $\perp$.

As discussed above, the denotational semantics of commands has type:

$$[\![c]\!] : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$$

The definition is given in Figure 3; it makes use of two auxiliary definitions. We first define:

$$\mathrm{Ass} : \mathrm{Mem}_{\xi\perp} \times \mathbb{N}_{\xi\perp} \times \mathbb{N}_{\xi\perp} \to \mathbb{N}_{\xi\perp}$$

by setting $\mathrm{Ass}(m, x, y) = m[x \mapsto y]$ if $x \in \mathrm{dom}(m)$ and $= \xi$, otherwise, for $m \in \mathrm{Mem}$, $x, y \in \mathbb{N}$, and then using the function extension associated to the "error + nontermination" monad. Second, we define

$$\mathrm{Cond}(p, \theta, \theta') : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$$

for any $p : \mathrm{Mem} \times W \to \mathbb{B}_{\xi\perp}$ and $\theta, \theta' : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, by:

$$\mathrm{Cond}(p, \theta, \theta')(\zeta) = \{\zeta' \mid \zeta'|_{W_{\zeta,\mathtt{tt}}} \in \theta(\zeta)|_{W_{\zeta,\mathtt{tt}}}, \zeta'|_{W_{\zeta,\mathtt{ff}}} \in \theta'(\zeta)|_{W_{\zeta,\mathtt{ff}}},$$
$$\zeta'(W_{\zeta,\xi}) \subseteq \{\xi\}, \text{ and } \zeta'(W_{\zeta,\perp}) \subseteq \{\perp\}\}$$

where $W_{\zeta,t} =_{\mathrm{def}} \{w \mid p(\zeta(w), w) = t\}$, for $t \in \mathbb{B}_{\xi\perp}$, and we apply restriction elementwise to sets of functions.

Example 4.1 In this example, we demonstrate our low-level denotational semantics. Consider the command:

$$c_5 = l'_{\mathtt{nat}} := l_{\mathtt{nat}}; (!l'_{\mathtt{nat}}) := 1; l'_{\mathtt{nat}} := 0$$

This command stores the address of location l at location l', then reads the contents of location l' (the address of l) and writes 1 at this address, and finally resets the memory at location l' to 0. Because of this manipulation of memory locations, this command is not the direct translation of a high-level command.

Letting:

$$s_{i,j} = (l \mapsto i, l' \mapsto j) \qquad \zeta_{i,j} = w \mapsto w \cdot s_{i,j} \qquad \zeta'_i = w \mapsto w \cdot (l \mapsto i, l' \mapsto w(l))$$

we have:

$$[\![l'_{\mathtt{nat}} := l_{\mathtt{nat}}]\!](\zeta_{i,j}) = \{\zeta'_i\}\!\downarrow$$

Note that $\zeta_{i,j}$ is a store projection, but ζ'_i is not. We also have:

$$[\![(!l'_{\mathtt{nat}}) := 1]\!](\zeta'_i) = \{\zeta'_1\}\!\downarrow \qquad [\![l'_{\mathtt{nat}} := 0]\!](\zeta'_1) = \{\zeta_{1,0}\}\!\downarrow$$

In sum, we have:

$$[\![c_5]\!](\zeta_{i,j}) = \{\zeta_{1,0}\}\!\downarrow$$

□

Looking at the type of the semantics

$$[\![c]\!] : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$$

one may be concerned that there is no apparent relation between the layouts used in the input to $[\![c]\!]$ and those in its output. However, we note that the semantics could be made parametric. For every $W' \subseteq W$, replace W by W' in the definition of $[\![c]\!]$ to obtain:

$$[\![c]\!]_{W'} : \mathrm{Mem}^{W'}_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^{W'}_{\xi\perp})$$

There is then a naturality property, that the following diagram commutes for all $W'' \subseteq W' \subseteq W$:

$$\begin{array}{ccc} \mathrm{Mem}^{W'}_{\xi\perp} & \xrightarrow{[\![c]\!]_{W'}} & \mathcal{H}(\mathrm{Mem}^{W'}_{\xi\perp}) \\ \Big\downarrow{\scriptstyle \mathrm{Mem}^{\iota}_{\xi\perp}} & & \Big\downarrow{\scriptstyle \mathcal{H}(\mathrm{Mem}^{\iota}_{\xi\perp})} \\ \mathrm{Mem}^{W''}_{\xi\perp} & \xrightarrow[[\![c]\!]_{W''}]{} & \mathcal{H}(\mathrm{Mem}^{W''}_{\xi\perp}) \end{array}$$

where $\iota : W'' \subseteq W'$ is the inclusion map. Taking $W' = W$ and W'' a singleton yields the expected relation between input and output: the value of a random variable in the output at a layout depends only on the value of the input random variable at that layout. The naturality property suggests re-working the low level denotational semantics in the category of presheaves over sets of layouts, and this may prove illuminating (see [12] for relevant background).

4.3 Operational semantics

As a counterpart to the denotational semantics, we give a deterministic operational semantics using oracles to make choices. The oracles are elements of the set Ω of infinite lists of tokens L (for "left") and R (for "right"). A low-level state σ is:

- a triple $\langle c, m, \pi\rangle$ of a command c, a memory m, and an oracle π; or
- a pair $\langle m, \pi\rangle$ of a memory m and an oracle π; or
- the error element ξ.

$$\frac{[\![e]\!]^w_m \in \mathrm{dom}(m) \text{ and } [\![e']\!]^w_m \neq \xi}{w \models \langle e := e', m, \pi\rangle \to \langle m[[\![e]\!]^w_m \mapsto [\![e']\!]^w_m], \pi\rangle} \qquad \frac{[\![e]\!]^w_m \notin \mathrm{dom}(m) \text{ or } [\![e']\!]^w_m = \xi}{w \models \langle e := e', m, \pi\rangle \to \xi}$$

$$\frac{[\![b]\!]^w_m = \mathrm{tt}}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \langle c, m, \pi\rangle}$$

$$\frac{[\![b]\!]^w_m = \mathrm{ff}}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \langle c', m, \pi\rangle} \qquad \frac{[\![b]\!]^w_m = \xi}{w \models \langle \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c', m, \pi\rangle \to \xi}$$

$$w \models \langle \mathsf{skip}, m, \pi\rangle \to \langle m, \pi\rangle \qquad \frac{w \models \langle c, m, \pi\rangle \to \langle c', m', \pi'\rangle}{w \models \langle c; c'', m, \pi\rangle \to \langle c'; c'', m', \pi'\rangle}$$

$$\frac{w \models \langle c, m, \pi\rangle \to \langle m', \pi'\rangle}{w \models \langle c; c'', m, \pi\rangle \to \langle c'', m', \pi'\rangle} \qquad \frac{w \models \langle c, m, \pi\rangle \to \xi}{w \models \langle c; c'' m, \pi\rangle \to \xi}$$

$$w \models \langle c + c', m, L\pi\rangle \to \langle c, m, \pi\rangle \qquad w \models \langle c + c', m, R\pi\rangle \to \langle c', m, \pi\rangle$$

$$\frac{[\![b]\!]^w_m = \mathrm{ff}}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \langle m, \pi\rangle}$$

$$\frac{[\![b]\!]^w_m = \mathrm{tt}}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \langle c; \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle} \qquad \frac{[\![b]\!]^w_m = \xi}{w \models \langle \mathtt{while}\, b\, \mathtt{do}\, c, m, \pi\rangle \to \xi}$$

Fig. 4. Low-level operational semantics

Transitions are given relative to a layout, so we write:

$$w \models \sigma \to \sigma'$$

The rules are given in Figure 4. This semantics is deterministic for each choice of layout. We write $w \models \sigma \Rightarrow \sigma'$ for the transitive closure of the transition relation (for a given layout).

Example 4.2 Consider the command c_4 introduced in Section 4.2.2, with added parentheses for disambiguation:

$$c_4 \;=\; (1{:=}1) \;+\; ((2{:=}1) \;+\; ((3{:=}1) \;+\; ((4{:=}1))))$$

We have:

$$\begin{array}{llll}
w_1 \models \langle c_4, w_1{\cdot}s_k, L\pi\rangle & \to \langle w_1{\cdot}s_1, \pi\rangle & w_j \models \langle c_4, w_j{\cdot}s_k, L\pi\rangle & \to \xi \ (j \neq 1) \\
w_2 \models \langle c_4, w_2{\cdot}s_k, RL\pi\rangle & \Rightarrow \langle w_2{\cdot}s_1, \pi\rangle & w_j \models \langle c_4, w_j{\cdot}s_k, RL\pi\rangle & \Rightarrow \xi \ (j \neq 2) \\
w_3 \models \langle c_4, w_3{\cdot}s_k, RRL\pi\rangle & \Rightarrow \langle w_3{\cdot}s_1, \pi\rangle & w_j \models \langle c_4, w_j{\cdot}s_k, RRL\pi\rangle & \Rightarrow \xi \ (j \neq 3) \\
w_4 \models \langle c_4, w_4{\cdot}s_k, RRR\pi\rangle & \Rightarrow \langle w_4{\cdot}s_1, \pi\rangle & w_j \models \langle c_4, w_j{\cdot}s_k, RRR\pi\rangle & \Rightarrow \xi \ (j \neq 4)
\end{array}$$

□

Using the operational semantics, we can define an evaluation function:

$$\mathrm{Eval} : \mathrm{Com} \times W \times \mathrm{Mem} \times \Omega \to \mathrm{Mem}_{\xi\perp}$$

by:

$$\mathrm{Eval}(c, w, m, \pi) = \begin{cases} m' & (w \models \langle c, m, \pi\rangle \Rightarrow \langle m', \pi'\rangle) \\ \xi & (w \models \langle c, m, \pi\rangle \Rightarrow \xi) \\ \perp & (\text{otherwise}) \end{cases}$$

We then define

$$\mathrm{Eval}_{\mathrm{ran}} : \mathrm{Com} \times \mathrm{Mem}_{\xi\perp}^{W} \to \Omega \to \mathrm{Mem}_{\xi\perp}^{W}$$

by:

$$\mathrm{Eval}_{\mathrm{ran}}(c, \zeta)(\pi)(w) = \begin{cases} \mathrm{Eval}(w, c, \zeta(w), \pi) & (\zeta(w) \in \mathrm{Mem}) \\ \zeta(w) & (\text{otherwise}) \end{cases}$$

Making use of the image functional $\mathrm{Im}_X : X^{\Omega} \to \mathcal{P}(X)$, where $\mathrm{Im}_X(f) = f(\Omega)$, we can state the consistency of the operational and denotational semantics:

Proposition 4.3 (Operational/denotational consistency) *For c a command and ζ a function in $\mathrm{Mem}_{\xi\perp}^{W}$, we have:*

$$[\![c]\!](\zeta) = \mathrm{Im}_{\mathrm{Mem}_{\xi\perp}^{W}}(\mathrm{Eval}_{\mathrm{ran}}(c, \zeta))\downarrow$$

The evaluation function yields operational correlates of the other possible denotational semantics discussed in Section 4.2.2, similarly, using image or induced distribution functionals. For example, for the first of those semantics, by currying Eval and composing, one obtains:

$$\mathrm{Com} \times W \times \mathrm{Mem} \xrightarrow{\mathrm{curry(Eval)}} \mathrm{Mem}_{\xi\perp}^{\Omega} \xrightarrow{\mathrm{Im}_{\mathrm{Mem}_{\xi\perp}}} \mathcal{P}(\mathrm{Mem}_{\xi\perp})$$

Using such operational correlates, one can verify operational versions of the assertions made in Section 4.2.2 about the inadequacies of those semantics.

4.4 Implementation relations and equivalences

Much as in the high-level language, we define a contextual implementation relation and a simulation relation for the low-level language. The low-level definitions refer to layouts, and in some cases include conditions on induced probabilities.

4.4.1 Contextual pre-order

Again, the contextual pre-order $c \sqsubseteq_L c'$ may be interpreted as saying that c "refines" (or "implements") c', in the sense that the publicly observable outcomes that c can produce are a subset of those that c' permits, in every public context. In comparison with definition for the high-level language, however, c and c' are not applied to an arbitrary initial store but rather to a function from layouts to memories (extended with "error + nontermination"), and they produce sets of such functions. We restrict attention to argument functions induced by stores, in the sense that they are store projections of the form $w \mapsto w{\cdot}s$. Thus, let $f = [\![C[c]]\!]$ and $f' = [\![C[c']]\!]$ for an arbitrary public context C, and let s be a store; then (roughly) for every ζ in $f(w \mapsto w{\cdot}s)$ there exists ζ' in $f'(w \mapsto w{\cdot}s)$ such that, for any w, $\zeta(w)$ and $\zeta'(w)$ coincide on public locations.

The treatment of error and nontermination introduces a further complication. Specifically, we allow that ζ produces an error or diverges with sufficient probability ($\geq \delta$), and that ζ' produces an error with sufficient probability ($\geq \delta$), as an alternative to coinciding on public locations.

Therefore, we define $\sqsubseteq_L$ and some auxiliary notation and relations:

- Set PubMem $=_{\text{def}} \mathbb{N}^{\text{ran}(w_p)}$. Then, for any memory m, let $m_L \in \text{PubMem}$ be the restriction of m to $\text{ran}(w_p)$, extending the notation to $\text{Mem}_{\xi\perp}$ as usual.
- For any $\zeta \in \text{Mem}^W_{\xi\perp}$, we define $\zeta_L \in \text{PubMem}^W_{\xi\perp}$ by setting $\zeta_L(w) = \zeta(w)_L$.
- For $X, Y \in \mathcal{H}(\text{Mem}^W_{\xi\perp})$, we write that $X \leq_L Y$ when, for every $\zeta \in X$, there exists $\zeta' \in Y$ such that:
  · $\zeta_L \leq \zeta'_L$, or
  · $P(\zeta(w) \in \{\xi, \perp\}) \geq \delta$ and $P(\zeta'(w) = \xi) \geq \delta$.
- For $f, f' \in \text{Mem}^W_{\xi\perp} \to \mathcal{H}(\text{Mem}^W_{\xi\perp})$, we write $f \leq_L f'$ when, for all $s \in S$, we have:

$$f(w \mapsto w{\cdot}s) \leq_L f'(w \mapsto w{\cdot}s)$$

- Finally, we write $c \sqsubseteq_L c'$ when, for every public command context C, $[\![C[c]]\!] \leq_L [\![C[c']]\!]$.

4.4.2 Simulation

As in the high-level language, we introduce a simulation relation $\preceq$. This relation works only on commands whose outcomes on inputs that are store projections are themselves store projections; nevertheless, simulation remains a useful tool for proofs.

We define $\varpi : S_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ by:

$$\begin{aligned}\varpi(\perp) &= \{w \mapsto \perp\}\downarrow \\ \varpi(s) &= \{w \mapsto w{\cdot}s\}\downarrow \\ \varpi(\xi) &= \{\zeta | P(\zeta(w) = \xi) \geq \delta\}\downarrow\end{aligned}$$

For every $X \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, we say that X is a store projection set when there exists $Y \in \mathcal{H}(S_{\xi\perp})$ such that

$$\varpi(Y \setminus \{\xi\})\downarrow \subseteq X \subseteq \varpi(Y)\downarrow$$

and

$$\xi \in Y \Rightarrow \exists \zeta \in X.\, P(\zeta(w) = \xi) \geq \delta$$

In that case, we write $\chi(X) = Y$ for the unique such Y; we have $s \in Y$ if, and only if, $w \mapsto w \cdot s \in X$ and $\xi \in Y$ if, and only if, $\exists \zeta \in X, P(\zeta(w) = \xi) \geq \delta$. (The uniqueness of Y depends on the assumption that $\delta > 0$.)

The $\leq_L$ relation restricted to store projection sets has a pleasant characterization. The notation $-_L$ extends from S to $S_{\xi\perp}$, so that $\perp_L = \perp$ and $\xi_L = \xi$; with that, for any X in $\mathcal{H}(S_{\xi\perp})$, define X_L in $\mathcal{H}(S_{L\xi\perp})$ to be $\{s_L \mid s \in X\}$.

Fact 4.4 *Let X and Y be store projection sets. Then:*

$$X \leq_L Y \quad \equiv \quad \chi(X)_L \leq \chi(Y)_L$$

Much as in the high-level language, we extend relations R over $\mathcal{H}(S_{H\xi\perp})$ to relations $R^\times$ over $\mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$. First we extend $-_s$ to $\mathcal{H}(S_{\xi\perp})$ as follows: for $X \in \mathcal{H}(S_{\xi\perp})$ and $s \in S_L$, we let $X_s \in \mathcal{H}(S_{H\xi\perp})$ be $(X \setminus \{\xi\})_s \cup \{\xi \mid \xi \in X\}$. Then, given a relation R over $\mathcal{H}(S_{H\xi\perp})$, we first extend it to a relation R^+ over $\mathcal{H}(S_{\xi\perp})$ by setting

$$\begin{aligned}X R^+ Y \equiv_{\mathrm{def}}\ & (\xi \in X \Rightarrow \xi \in Y) \wedge \\ & \forall s \in S_L.\, ((X_s \setminus \xi) \neq \{\perp\} \Rightarrow (Y_s \setminus \xi) \neq \{\perp\}) \wedge X_s R Y_s\end{aligned}$$

for $X, Y \in \mathcal{H}(S_{\xi\perp})$ and then define $R^\times$ by setting:

$$X R^\times Y \equiv_{\mathrm{def}} X \text{ and } Y \text{ are store projection sets} \wedge \chi(X) R^+ \chi(Y)$$

for $X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$. (Note that if $R \subseteq \mathcal{H}(S_{H\perp})$, then the high- and low-level definitions of R^+ coincide.)

If R is closed under increasing ω-sups (respectively, is right-closed under $\leq$, is closed under binary unions) the same holds for R^+, and then for $R^\times$ (with $\leq$ restricted to store projection sets). If R is reflexive, then R^+ is and $R^\times$ is reflexive on store projection sets. We also have, much as before, that, for $X, Y \in \mathcal{H}(S_{\xi\perp})$, if

XR^+Y then $X_L \leq Y_L$. It then follows from Fact 4.4 that, for $X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$, if $XR^\times Y$ then $X \leq_L Y$.

For any $f, f' : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ and relation R over $\mathcal{H}(S_{H\perp})$ we write that $f \preceq_R f'$ when:

$$\forall X, Y \in \mathcal{H}(\mathrm{Mem}^W_{\xi\perp}).\, XR^\times Y \Rightarrow f^\dagger(X) R^\times f'^\dagger(Y)$$

Finally, we write that $f \preceq f'$ if $f \preceq_R f'$ for some reflexive R closed under increasing ω-sups, right-closed under $\leq$, and closed under binary unions.

4.4.3 Contextual pre-order vs. simulation

The contextual pre-order coincides with the simulation relation, but only for commands whose semantics sends store projections to store projection sets. Formally, we say that a given function $f : \mathrm{Mem}^W_{\xi\perp} \to \mathcal{H}(\mathrm{Mem}^W_{\xi\perp})$ preserves store projections if, for every $s \in S$, $f(w \mapsto w{\cdot}s)$ is a store projection set. The coincidence remains quite useful despite this restriction, which in particular is not an impediment to our overall goal of relating the low-level language to the high-level language.

Theorem 4.5 *Let c and c' be two commands of the low-level language such that $[\![c]\!]$ and $[\![c']\!]$ preserve store projections. Then $c \sqsubseteq_L c'$ holds if and only if $[\![c]\!] \preceq [\![c']\!]$ does.*

Example 4.6 Suppose that there is only one private location, and consider the two commands:

$$c_4 = (1{:=}1) \;+\; (2{:=}1) \;+\; (3{:=}1) \;+\; (4{:=}1) \qquad c_6 = (1{:=}1); (2{:=}1)$$

As seen above, we have that $[\![c_4]\!](\zeta_i) = \{\zeta^1_\xi, \zeta^2_\xi, \zeta^3_\xi, \zeta^4_\xi\}{\downarrow}$. We also have that $[\![c_6]\!](\zeta_i) = \{w \mapsto \xi\}{\downarrow}$. Since $P(\zeta^i_\xi(w) = \xi) \geq \delta$, we can verify that c_4 and c_6 are equivalent. (Thus, a nondeterministic guess is no better than failure.) □

5 High and low

In this section we investigate the relation between the high-level language and the low-level language. Specifically, we define a simple translation from the high-level language to the low-level language, then we study its properties.

We define the compilation of high-level commands c (expressions e, boolean expressions b) to low-level commands $c^\downarrow$ (expressions $e^\downarrow$ and boolean expressions $b^\downarrow$) by setting: $(!l_{\mathtt{loc}})^\downarrow = !l_{\mathtt{nat}}$, $(l_{\mathtt{loc}} := e)^\downarrow = l_{\mathtt{nat}} := e^\downarrow$, and proceeding homomorphically in all other cases (e.g., $(e + e')^\downarrow = e^\downarrow + e'^\downarrow$). Crucially, this compilation function, which is otherwise trivial, transforms high-level memory access to low-level memory access.

Lemma 5.1 *Let c be a high-level command. Then $[\![c^\downarrow]\!]$ preserves store projections.*

Theorem 5.2 relates the simulation relations of the two languages. It states that a high-level command c simulates another high-level command c, with respect to

all public contexts of the high-level language, if and only if the compilation of c simulates the compilation of c', with respect to all public contexts of the low-level language.

Theorem 5.2 *Let c and c' be two high-level commands. Then $[\![c]\!] \preceq [\![c']\!]$ holds if and only if $[\![c^{\downarrow}]\!] \preceq [\![c'^{\downarrow}]\!]$ does.*

Our main theorem, Theorem 5.3, follows from Theorem 5.2, the two previous theorems, and the lemma. Theorem 5.3 is analogous to Theorem 5.2, but refers to the contextual pre-orders: a high-level command c implements another high-level command c', with respect to all public contexts of the high-level language, if and only if the compilation of c implements the compilation of c', with respect to all public contexts of the low-level language.

Theorem 5.3 (Main theorem) *Let c and c' be two high-level commands. Then $c \sqsubseteq_L c'$ holds if and only if $c^{\downarrow} \sqsubseteq_L c'^{\downarrow}$ does.*

Theorem 5.3 follows from Theorem 5.2, the two previous theorems, and the lemma. The low-level statement is defined in terms of the probability δ that depends on the distribution on memory layouts. When δ is close to 1, the statement indicates that, from the point of view of a public context (that is, an attacker), the compilation of c behaves like an implementation of the compilation of c'. This implementation relation holds despite the fact that the public context may access memory via natural-number addresses, and thereby (with some probability) read or write private data of the commands. The public context may behave adaptively, with memory access patterns chosen dynamically, for instance attempting to exploit correlations in the distribution of memory layouts. The public context may also give "unexpected" values to memory addresses, as in practical attacks; the theorem implies that such behavior is no worse at the low level than at the high level.

For example, for the commands c_0 and c_1 of Example 3.1, the theorem enables us to compare how their respective compilations behave, in an arbitrary public low-level context. Assuming that δ is close to 1, the theorem basically implies that a low-level attacker that may access memory via natural-number addresses cannot distinguish those compilations. Fundamentally, this property holds simply because the attacker can read or write the location h only with low probability.

6 Conclusion

A few recent papers investigate the formal properties of layout randomization, like ours [19,3,13,2]. They do not consider nondeterministic choice, and tend to reason operationally. However, the work of Jagadeesan et al. includes some semantic elements that partly encouraged our research; specifically, that work employs trace equivalence as a proof technique for contextual equivalence.

In this paper we develop a semantic approach to the study of layout randomization. Our work concerns nondeterministic languages, for which this approach

has proved valuable in reconciling probabilistic choice with nondeterministic choice. However, the approach is potentially more general. In particular, the study of concurrency with nondeterministic scheduling would be an attractive next step. Also, extending our work to higher-order computation presents an interesting challenge.

References

[1] M. Abadi and L. Lamport. The existence of refinement mappings. *TCS*, 82(2):253–284, 1991.

[2] M. Abadi and J. Planul. On layout randomization for arrays and functions. In *POST*, volume 7796 of *LNCS*, pages 167–185. Springer, 2013.

[3] M. Abadi and G. D. Plotkin. On protection by layout randomization. *ACM Transactions on Information and System Security*, 15(2):8:1–8:29, 2012.

[4] R. Canetti et al. Analyzing security protocols using time-bounded task-pioas. *Discrete Event Dynamic Systems*, 18(1):111–159, 2008.

[5] W. P. de Roever and K. Engelhardt. *Data Refinement: Model-oriented Proof Theories and their Comparison*, volume 46 of *Cambridge Tracts in Theo. Comp. Sci.* CUP, 1998.

[6] P. Druschel and L. L. Peterson. High-performance cross-domain data transfer. Technical Report TR 92-11, Department of Computer Science, The University of Arizona, 1992.

[7] Ú. Erlingsson. Low-level software security: Attacks and defenses. In *FOSAD IV Tutorial Lectures*, volume 4677 of *LNCS*, pages 92–134. Springer, 2007.

[8] S. Forrest et al. Building diverse computer systems. In *6th Workshop on Hot Topics in Operating Systems*, pages 67–72, 1997.

[9] G. Gierz et al. *Continuous lattices and domains*, volume 93 of *Encyclopaedia of mathematics and its applications*. CUP, 2003.

[10] S. Goldwasser and S. Micali. Probabilistic encryption. *JCSS*, 28:270–299, 1984.

[11] J. Goubault-Larrecq. Prevision domains and convex powercones. In *FoSSaCS*, volume 4962 of *LNCS*, pages 318–333. Springer, 2008.

[12] M. Jackson. *A sheaf theoretic approach to measure theory*. PhD thesis, U. Pitt., 2006.

[13] R. Jagadeesan et al. Local memory via layout randomization. In *Proc. of the 24th CSFS*, pages 161–174, 2011.

[14] N. Klarlund and F. B. Schneider. Proving nondeterministically specified safety properties using progress measures. *Information and Computation*, 107(1):151–170, 1993.

[15] P. Lincoln et al. A probabilistic poly-time framework for protocol analysis. In *Proceedings of the Fifth ACM Conference on Computer and Communications Security*, pages 112–121, 1998.

[16] M. W. Mislove. On combining probability and nondeterminism. *ENTCS*, 162:261–265, 2006.

[17] J. C. Mitchell et al. A probabilistic polynomial-time process calculus for the analysis of cryptographic protocols. *TCS*, 353(1-3):118–164, 2006.

[18] PaX Project. The PaX project, 2004. http://pax.grsecurity.net/.

[19] R. Pucella and F. B. Schneider. Independence from obfuscation: A semantic framework for diversity. *Journal of Computer Security*, 18(5):701–749, 2010.

[20] A. Sabelfeld and D. Sands. Probabilistic noninterference for multi-threaded programs. In *CSFW*, pages 200–214, 2000.

[21] R. Tix et al. Semantic domains for combining probability and non-determinism. *ENTCS*, 222:3–99, 2009.