Layout Randomization and Nondeterminism

Martín Abadi Jérémy Planul Gordon Plotkin

PrakashFest, Oxford, May 2014

Hoping to work with Prakash: Choquet capacities

Josée, Vineet, Radha, and Prakash, CONCUR '02:
Weak Bisimulation is Sound and Complete for pCTL* .

- Superadditive Choquet capacity:

$$\mu(A \cup B) \leq \mu(A) + \mu(B) \quad (A, B \subseteq \Sigma, \text{disjoint})$$

- Given measures μ_i, their lower envelope:

$$\mu(A) =_{\text{def}} \min_i \mu_i(A)$$

  is a superadditive Choquet capacity.
- Can we do ordinary + probabilistic nondeterminism this way?

Hoping to work with Prakash: Choquet capacities

Josée, Vineet, Radha, and Prakash, CONCUR '02:
Weak Bisimulation is Sound and Complete for pCTL* .

- Superadditive Choquet capacity:

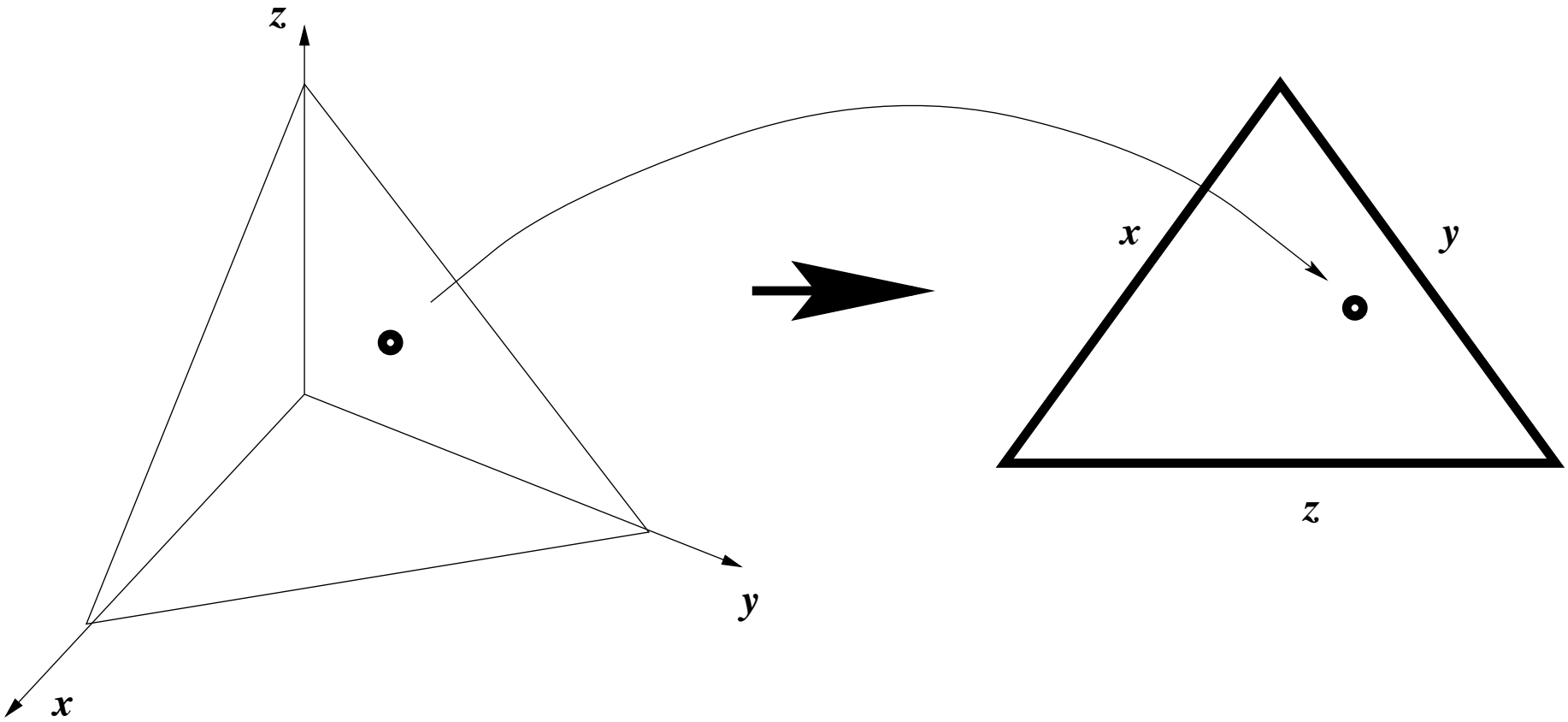
$$\mu(A \cup B) \leq \mu(A) + \mu(B) \quad (A, B \subseteq \Sigma, \text{disjoint})$$

- Given measures μ_i, their lower envelope:

$$\mu(A) =_{\text{def}} \min_i \mu_i(A)$$

  is a superadditive Choquet capacity.
- Can we do ordinary + probabilistic nondeterminism this way?
- Sadly, this map is many-to-one from convex sets of measures to capacities, even when $|\Sigma| = 3$.

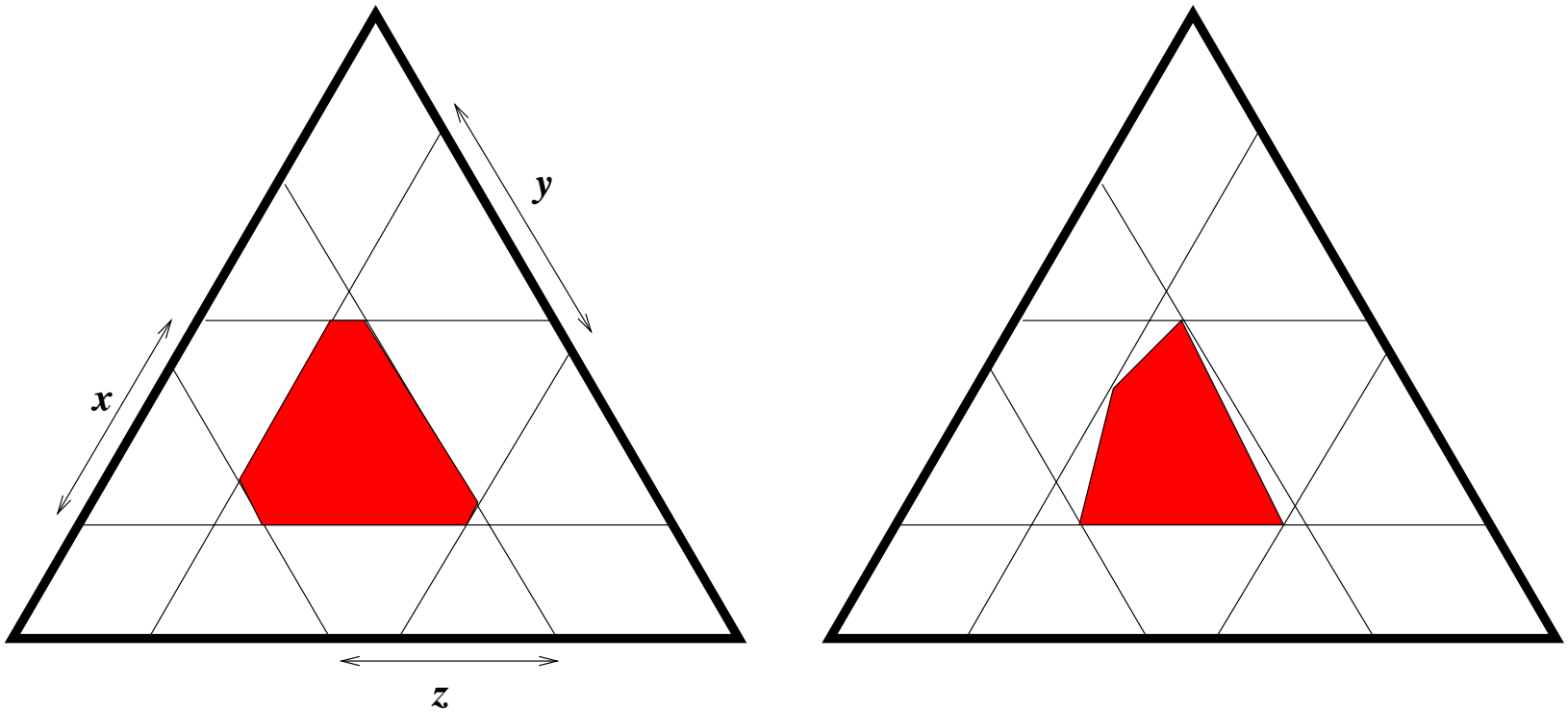
Barycentric coordinates



Cartesian coordinates

Barycentric coordinates

Counterexamples in plenty (Cozman)



Trying to work with Prakash: quantum computation

A Language and Type Theory for Quantum Computing

Prakash Panangaden*
School of Computer Science
McGill University
Montreal, Quebec, Canada

Gordon D. Plotkin†
Department of Computer Science
University of Edinburgh
Edinburgh, Scotland, U. K.

February 28, 2003

Abstract

The abstract is very uncertain.

1 Introduction

Outline the nature of quantum comp as a web of interactions Our philosophy: evolution is fundamental, density ops, intervention operators are fundamental, no relativistic effects, entanglement, expressing unitaries; cite Peres and NC

Computation is dynamics. One prepares a dynamical system in some initial state - implicitly or explicitly containing the "input" - and lets it evolve. The "answer" is contained in the final state. Understanding quantum computation must, at its lowest level, involve understanding quantum evolution.

Normally in quantum mechanics understanding quantum evolution means understanding the generator of time evolution, i.e. the Hamiltonian operator. If H is the Hamiltonian operator for a system then the evolution of states is governed by the *unitary* operator e^{-iHt}. Typically this is cast as a differential equation, the Schrodinger equation. In typical text books one spends a significant amount of time solving the Schrodinger equation for a variety of Hamiltonians.

In quantum computation textbooks [NC00] one takes various unitary operations on states of collections two-state systems (qubits) as given. These two-state systems, or qubits, are conceived of as separate carriers of information, just like classical bits. Physically they are generally realized as spin-states of spin $\frac{1}{2}$ particles or as polarization states of photons. What is very different about qubits versus bits is that (i) a qubit may exist in a superposed state; i.e. in a linear combination of its two basic states and (ii) states of different qubits may be *entangled* i.e. correlated in nontrivial ways.

Algorithms are developed by combining the primitive unitary operators and occasionally by *making measurements*. However, the primitive unitaries do not all just act on one qubit at a time. In any presentation of the primitives, there is at least one primitive that involves multiple (usually

Somewhat working with Prakash: Markov processes

Approximating Markov Processes By Averaging

PHILIPPE CHAPUT
McGill University
VINCENT DANOS
University of Edinburgh
PRAKASH PANANGADEN
McGill University
GORDON PLOTKIN
University of Edinburgh

October 14, 2013

Abstract

Normally, one thinks of probabilistic transition systems as taking an initial probability distribution over the state space into a new probability distribution representing the system after a transition. We, however, take a dual view of Markov processes as transformers of bounded measurable functions. This is very much in the same spirit as a "predicate-transformer" view, which is dual to the state-transformer view of transition systems.

We redevelop the theory of labelled Markov processes from this view point, in particular we explore approximation theory. We obtain three main results:
(i) It is possible to define bisimulation on general measure spaces and show that it is an equivalence relation. The logical characterization of bisimulation can be done straightforwardly and generally. (ii) A new and flexible approach to approximation based on averaging can be given. This vastly generalizes and streamlines the idea of using conditional expectations to compute approximations. (iii) We show that there is a minimal process bisimulation-equivalent to a given process, and this minimal process is obtained as the limit of the finite approximants.

Low-Level Attacks and Protection

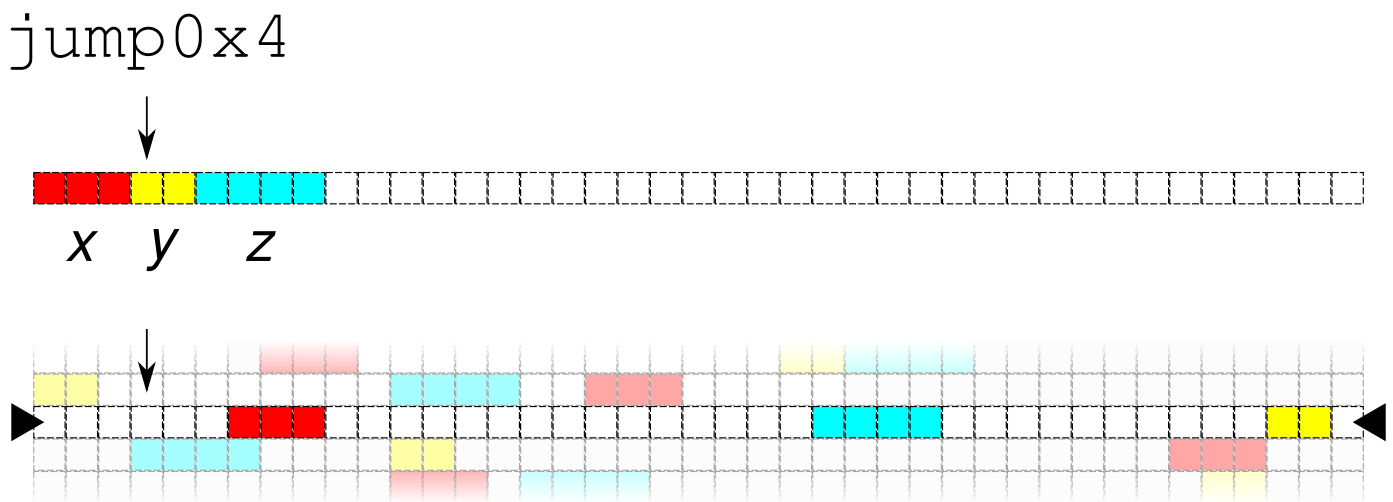
- Many attack techniques:
  - Buffer overflows
  - Exception overwrites
  - return-to-libc
  - jump-to-libc
  - use-after-free attacks
- Often with knowledge and control of the heap

- Many defenses:
  - Stack canaries
  - Safe exception handling
  - NX (No eXecute) data
  - Layout randomization
- Useful mitigations
- But not necessarily perfect in a precise sense
- Nor all well understood

Layout randomization

Runtime attacks often depend on addresses (e.g. jump to `libc`).

- Let us randomize the addresses!
  - Considered for data at least since the rise of large virtual address spaces (e.g., [Druschel & Peterson, 1992] on fbufs).
  - Now present in Linux (PaX), Windows, Mac OS X, iOS, Android (4.0).

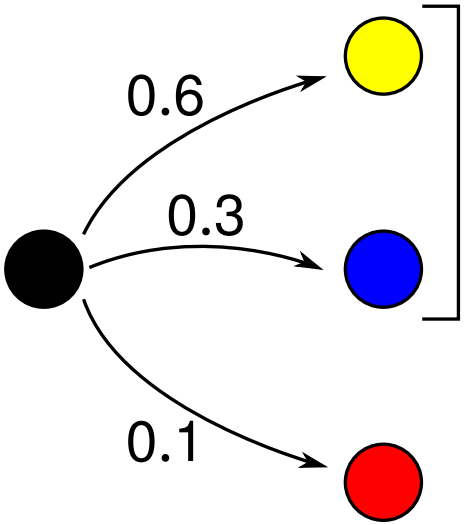


A theory of layout randomization

- Abadi and Plotkin did a first formalization, using operational methods.
- Further work [Jagadeesan et al., Abadi et al.] concerned memories containing functions and arrays.
- Here we consider commands that make nondeterministic choices (as a first step towards parallelism).
- We use denotational methods rather than operational ones (for interest, but not only).
- We use random variables for probability rather than distributions, as events are not independent.

Probabilities and nondeterminism

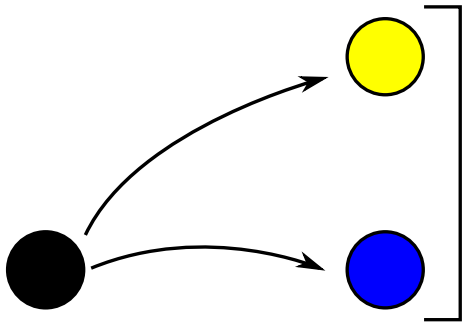
Probabilities:



Not red with ≥ 0.9 probability

- Probabilistic cryptographic schemes
- Differential privacy
- Low-level software protection (Layout randomization)

Nondeterminism:



Never red

- Network communications
- Nondeterministic scheduling
- User interaction

Probabilities ⟷ Nondeterminism: Game theory

- Player 1 chooses a bit (b_1) by flipping a coin
- Player 2 chooses a bit (b_2) nondeterministically
- Player 1 wins if he guesses right:

$$b_1 = b_2$$

- The program:

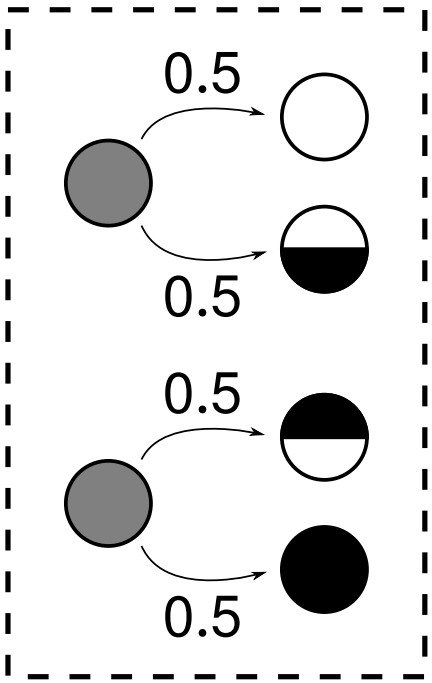
$$b_1 := 0 +_{\frac{1}{2}} 1; \left(b_2 := 0 \ \text{ OR } \ b_2 := 1\right)$$

Probabilities ⟷ Nondeterminism: Game theory

$$b_1 := 0 +_{\frac{1}{2}} 1; (b_2 := 0 \text{ OR } b_2 := 1)$$

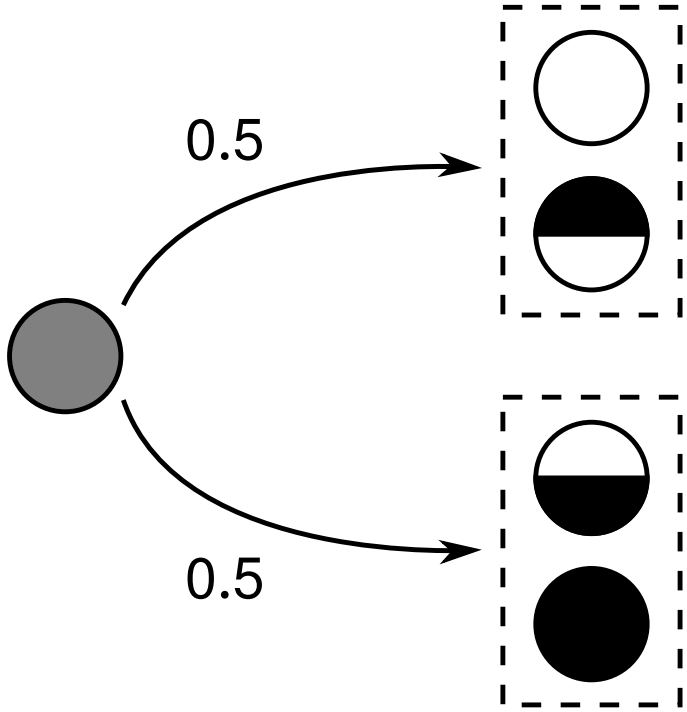
What should its semantics be?

- a set of distributions on outcomes



- Player I always wins with probability 1/2.

- a distribution on sets of outcomes

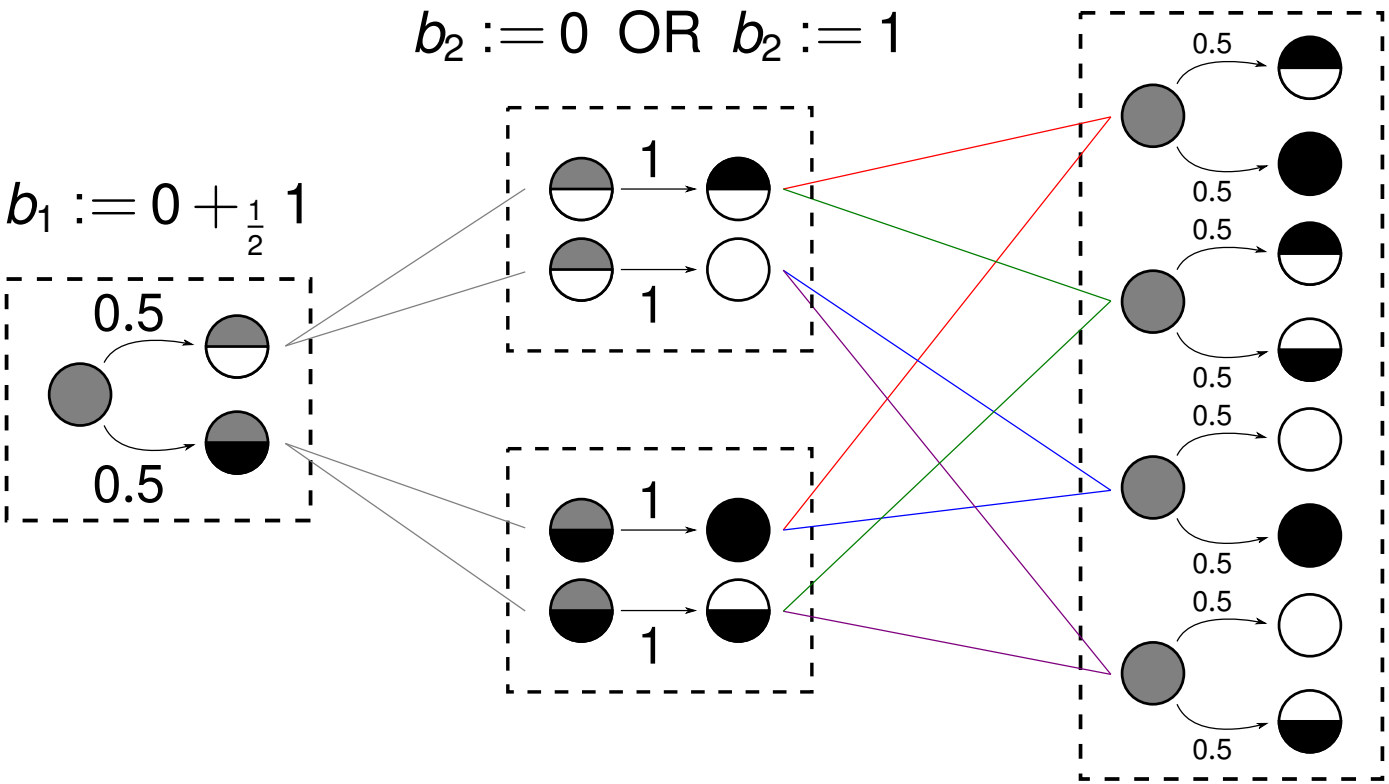


- Player I can win or lose.

Probabilities ⟷ Nondeterminism: Game theory

$$b_1 := 0 +_{\frac{1}{2}} 1; (b_2 := 0 \text{ OR } b_2 := 1)$$

The obvious semantics for sets of distributions on outcomes does not compose as expected:



Probabilities ⟷ Nondeterminism: Algebra

- Resolving probabilistic choice first:
  - Distribute nondeterministic choice over probabilistic choice:

$$x \cup (y +_p z) = (x \cup y) +_p (x \cup z)$$

  - One can prove that for all p, q: $x +_p y = x +_q y$ so the $+_p$ become a second semilattice, distributing over the first one.
  - Free Algebra over a set X: Non-empty finite $\cup$-closed sets of non-empty finite subsets of X.
- Resolving nondeterministic choice first:
  - Distribute probabilistic choice over nondeterministic choice:

$$x +_p (y \cup z) = (x +_p y) \cup (x +_p z)$$

  - Free Algebra over a set X: Non-empty finitely generated convex sets of distributions over X.
  - But gives the "wrong" semantics to our program.

Roadmap

- Define high-level and low-level languages, each with access to (the same) private and public locations.
  $\Rightarrow$ View the implementations as compilations.
- View attackers as (high- or low-level) "public" contexts: those with no access to private locations.
  $\Rightarrow$ Relate commands depending on their public behaviour in public contexts.
- Characterize the contextual relations using semantical simulations (a kind of logical relation)
  $\Rightarrow$ prove that high-level contextual relations correspond to low-level contextual relations (full abstraction):

$$\begin{array}{ccc} [\![c]\!] \preceq [\![c']\!] & \Leftrightarrow & \forall \text{high-level public } C[\,].\, [\![C[c]]\!] \leq_L [\![C[c']]\!] \\ \Updownarrow & & \\ [\![c^{\downarrow}]\!] \preceq [\![c'^{\downarrow}]\!] & \Leftrightarrow & \forall \text{low-level public } C[\,].\, [\![C[c^{\downarrow}]]\!] \leq_L [\![C[c'^{\downarrow}]]\!] \end{array}$$

High-level language

$$
\begin{array}{lcl}
e & ::= & k \mid !l_{\mathtt{loc}} \mid e + e \mid e * e \\
b & ::= & e \leq e \mid \neg b \mid \mathtt{true} \mid \mathtt{false} \mid b \vee b \mid b \wedge b \\
c & ::= & l_{\mathtt{loc}} := e \mid \mathtt{if}\, b\, \mathtt{then}\, c\, \mathtt{else}\, c \mid \mathsf{skip} \mid c; c \mid \\
 & & c + c \mid \mathtt{while}\, b\, \mathtt{do}\, c
\end{array}
$$

- Finitely many locations.
- They are divided into private (high) and public (low).
- store s: locations $\longrightarrow$ values
- Semantics $[\![c]\!]$: stores $\longrightarrow \mathscr{H}(\text{stores}_\perp)$

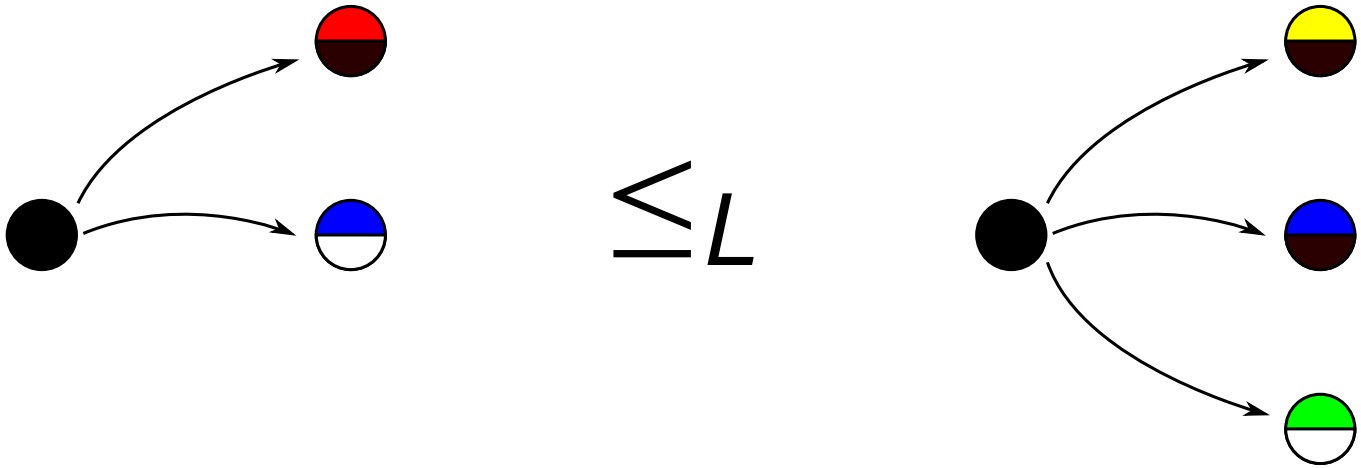
High-level language

$$
\begin{array}{lcl}
e & ::= & k \mid !l_{\mathtt{loc}} \mid e + e \mid e * e \\
b & ::= & e \leq e \mid \neg b \mid \mathtt{true} \mid \mathtt{false} \mid b \vee b \mid b \wedge b \\
c & ::= & l_{\mathtt{loc}} := e \mid \mathtt{if}\ b\ \mathtt{then}\ c\ \mathtt{else}\ c \mid \mathsf{skip} \mid c; c \mid \\
 & & c + c \mid \mathtt{while}\ b\ \mathtt{do}\ c
\end{array}
$$

- Finitely many locations.
- They are divided into private (high) and public (low).
- store s: locations $\longrightarrow$ values
- Semantics $[\![c]\!]$: stores $\longrightarrow \mathscr{H}(\text{stores}_\perp)$

Refinement relation (high-level)

- For $X, Y \in \mathscr{H}(\text{stores}_\perp)$, X refines Y if the public part of any store in X is the public part of a store in Y.



Formally:

$$X \leq_L Y \equiv \forall s \in \text{stores}.\, s_L \in X \Rightarrow s_L \in Y$$

- For commands, c refines c', written $c \leq_L c'$ when, for every input, the outputs of c refine those of c':

$$\forall s \in \text{stores}.\, [\![c]\!](s) \leq_L [\![c']\!](s)$$

Simulation (high-level)

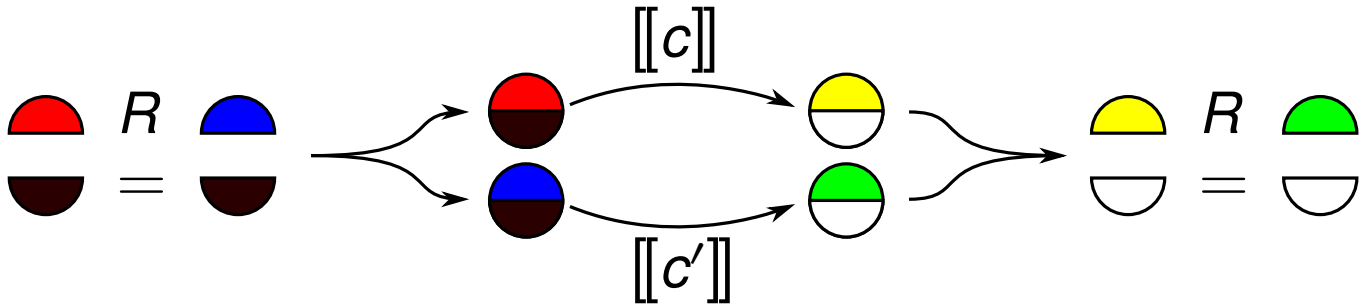
- Simulation relation $\preceq$ on commands such that

Theorem 1

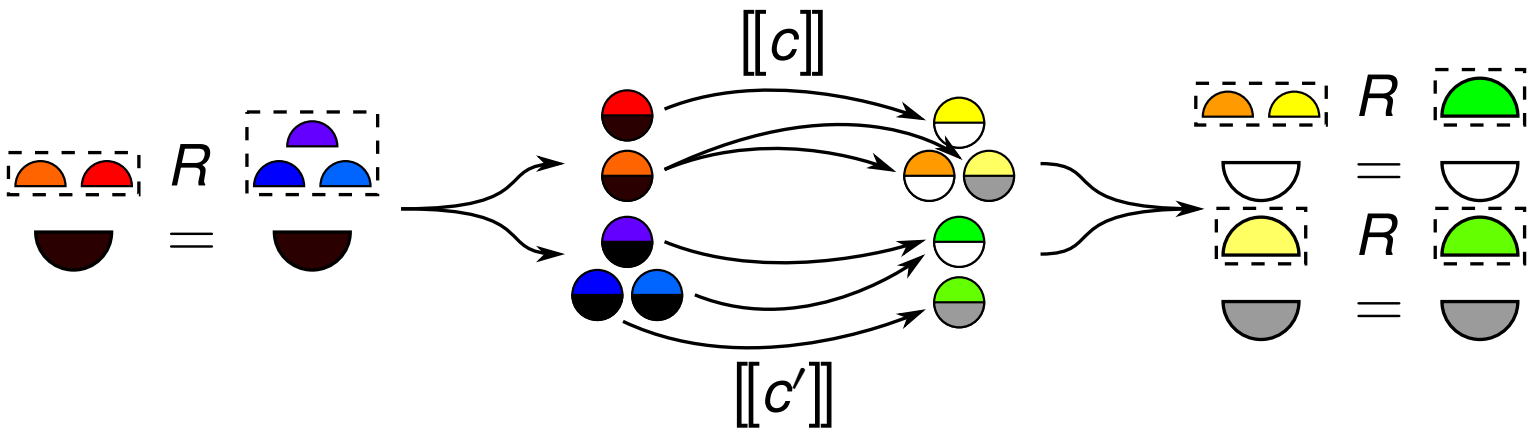
For all high-level commands c, c′, we have:

$$[\![c]\!] \preceq [\![c']\!] \iff \forall \textit{ high-level public } C[\,].\, [\![C[c]]\!] \leq_L [\![C[c']]\!]$$

- Simple without nondeterminism or nontermination:



Simulation (details)



- So begin with $R \subseteq \mathscr{H}(\text{private-stores}_\perp)^2$, subject to conditions
- Then construct $R^+ \subseteq \mathscr{H}(\text{stores}_\perp)^2$ from R
- Then

$$[[c]] \preceq [[c']] \equiv_{\text{def}} \forall X, Y \in \mathscr{H}(\text{stores}_\perp).\, X R^+ Y \Rightarrow [[c]](X) R^+ [[c']](Y)$$

Low-level memory model

- Memories ($m \in \mathrm{Mem}$) are partial mappings from the set of memory addresses Add $= \{1, \ldots, r\}$ to values.
- Memory layouts ($w \in \mathrm{Lay}$) are injective mappings from locations to the addresses Add.

$\Longrightarrow$ stores map to memories via layouts.

$$s \begin{bmatrix} l_1 \longmapsto \blacksquare \\ l_2 \longmapsto \blacksquare \\ l_3 \longmapsto \blacksquare \end{bmatrix} \qquad w \begin{bmatrix} l_1 \longmapsto 7 \\ l_2 \longmapsto 28 \\ l_3 \longmapsto 14 \end{bmatrix}$$

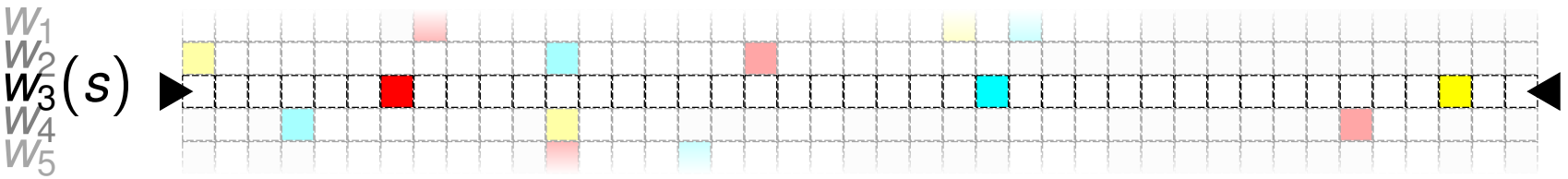
$w(s)$ 

Layout randomization

- We fix a public layout w_p.
- We randomize the private layout (using a distribution d)
- We rely on one value in our theorems, the minimum probability for a guessed private address to be unallocated:

$$\begin{aligned} \delta &= \min\{P(i \notin \mathrm{ran}(w)) \mid i \in \mathsf{Add} \setminus \mathrm{ran}(w_p)\} \\ &= 1 - |\mathsf{PriLoc}|/|\mathsf{Add} \setminus \mathrm{ran}(w_p)| \quad (d \text{ uniform}) \end{aligned}$$

$$s \begin{bmatrix} l_1 \mapsto \blacksquare \\ l_2 \mapsto \blacksquare \\ l_3 \mapsto \blacksquare \end{bmatrix} \qquad w_3 \begin{bmatrix} l_1 \mapsto 7 \\ l_2 \mapsto 28 \\ l_3 \mapsto 14 \end{bmatrix}$$



Low-level language

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

- We can compute over $l_{\mathtt{nat}}$, ex: $(x + l_{\mathtt{nat}}) :=_{\mathtt{nat}} (x + l'_{\mathtt{nat}})$.

Low-level language

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

- We can compute over $l_{\mathtt{nat}}$, ex: $(x + l_{\mathtt{nat}}) :=_{\mathtt{nat}} (x + l'_{\mathtt{nat}})$.

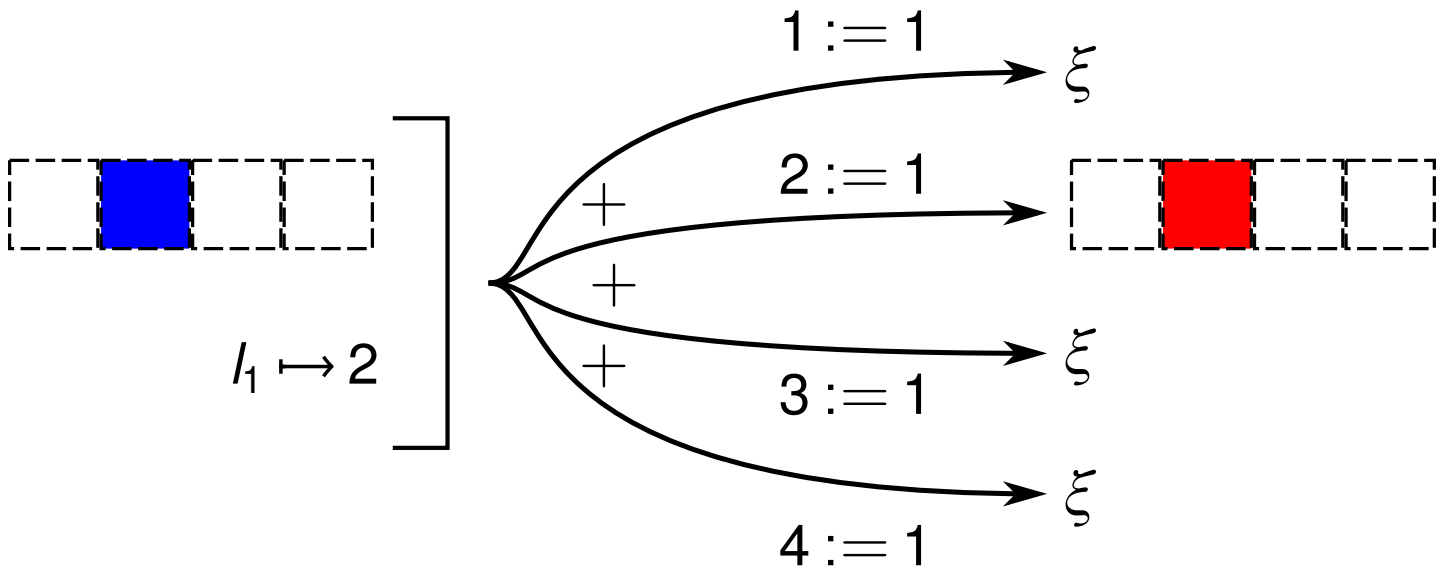
Low-level semantics

- The final memory depends on the initial one and the layout,
- We cannot prove security using a semantics of the form:

$$\mathrm{Mem} \times \mathrm{Lay} \to \mathscr{H}\left(\mathrm{Mem}_{\xi,\perp}\right)$$

- for consider the command:

$$(1:=1) + (2:=1) + (3:=1) + (4:=1)$$



Low-level semantics

- So we need to resolve probabilities after nondeterminism:
- However

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

  does not work as our events are not independent.
- For example, suppose there is just one private location l_1, and 1,2 are not public addresses. Then:
  - $1 := 1\,;\, 2 := 1$ should always give an error.
  - $1 := 1\,;\, 1 := 2$ should give an error or overwrite l_1

Low-level semantics

- Solution idea: replace distributions

$$d \in \mathscr{V}(\mathrm{Mem}_{\xi,\perp})$$

by random variables

$$\zeta : \mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp}$$

with layouts as the sample space, obtaining:

$$\mathrm{Mem} \to \mathscr{H}(\mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp})$$

which compose, using Kleisli structure on $\mathscr{H}$.

Low-level semantics

- Solution idea: replace distributions

$$d \in \mathscr{V}(\mathrm{Mem}_{\xi,\perp})$$

by random variables

$$\zeta : \mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp}$$

with layouts as the sample space, obtaining:

$$\mathrm{Mem} \to \mathscr{H}(\mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp})$$

- Sadly, these do not compose (cf intro) and we instead use:

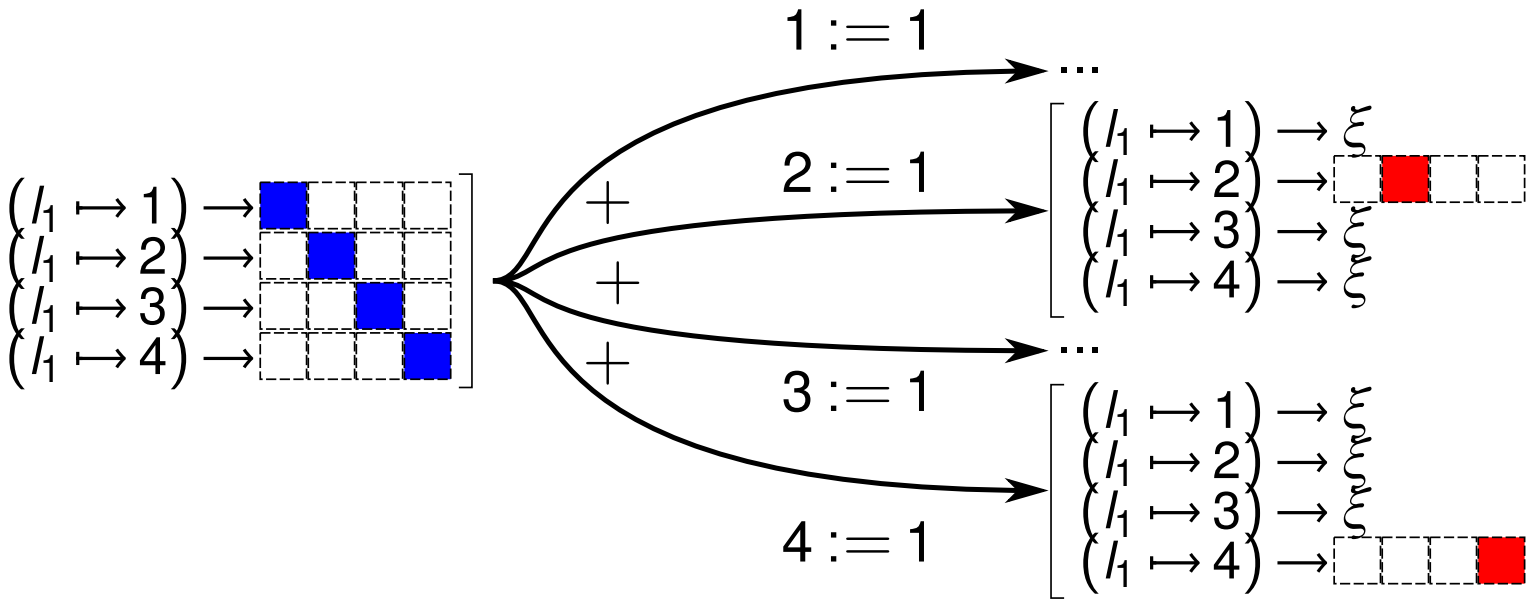
$$(\mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp}) \to \mathscr{H}(\mathrm{Lay} \to \mathrm{Mem}_{\xi,\perp})$$

which compose, using Kleisli structure on $\mathscr{H}$.

Low level semantics

- Example:

$$(1:=1) + (2:=1) + (3:=1) + (4:=1)$$



- So, as desired, always get an error with high probability.

Low-level semantics

$$[\![c]\!] : (\mathsf{Lay} \to \mathrm{Mem}_{\xi,\perp}) \to \mathscr{H}(\mathsf{Lay} \to \mathrm{Mem}_{\xi,\perp})$$

Most cases are straightforward:

$$\begin{array}{lcl}
[\![c + c']\!](\zeta) & = & [\![c]\!](\zeta) \cup [\![c']\!](\zeta) \\
[\![c; c']\!] & = & [\![c']\!]^\dagger \circ [\![c]\!] \\
[\![\mathsf{skip}]\!] & = & \eta \\
[\![e := e']\!](\zeta) & = & \begin{cases} \eta(\lambda w.\, m[[\![e]\!]^w_{\zeta(w)} \mapsto [\![e']\!]^w_{\zeta(w)}]) & \\ & (\text{if } [\![e]\!]^w_{\zeta(w)} \in \mathrm{dom}(m)) \\ \eta(\lambda w.\, \xi) & (\text{otherwise}) \end{cases} \\
[\![\texttt{if}\, b\, \texttt{then}\, c\, \texttt{else}\, c']\!] & = & \mathrm{Cond}([\![b]\!], [\![c]\!], [\![c']\!]) \\
[\![\texttt{while}\, b\, \texttt{do}\, c]\!] & = & \mu\theta.\mathrm{Cond}([\![b]\!], \theta^\dagger \circ [\![c]\!], \eta)
\end{array}$$

Example conditional semantics

$$[\![b]\!]\begin{pmatrix} w_1 \mapsto m_1 \\ w_2 \mapsto m_2 \\ w_3 \mapsto m_3 \end{pmatrix} = \left[\begin{array}{l} w_1 \mapsto \text{false} \\ w_2 \mapsto \text{true} \\ w_3 \mapsto \text{false} \end{array}\right]$$

$$[\![c]\!]\begin{pmatrix} w_1 \mapsto m_1 \\ w_2 \mapsto m_2 \\ w_3 \mapsto m_3 \end{pmatrix} = \left\{\begin{array}{l} w_1 \mapsto m_{c1} \\ w_2 \mapsto m_{c2} \\ w_3 \mapsto m_{c3} \end{array}, \begin{array}{l} w_1 \mapsto m'_{c1} \\ w_2 \mapsto m'_{c2} \\ w_3 \mapsto m'_{c3} \end{array}\right\}$$

$$[\![c']\!]\begin{pmatrix} w_1 \mapsto m_1 \\ w_2 \mapsto m_2 \\ w_3 \mapsto m_3 \end{pmatrix} = \left\{\begin{array}{l} w_1 \mapsto m_{c'1} \\ w_2 \mapsto m_{c'2} \\ w_3 \mapsto m_{c'3} \end{array}, \begin{array}{l} w_1 \mapsto m'_{c'1} \\ w_2 \mapsto m'_{c'2} \\ w_3 \mapsto m'_{c'3} \end{array}\right\}$$

$$[\![\texttt{if}\ b\ \texttt{then}\ c\ \texttt{else}\ c']\!]\begin{pmatrix} w_1 \mapsto m_1 \\ w_2 \mapsto m_2 \\ w_3 \mapsto m_3 \end{pmatrix} = \left\{\begin{array}{l} w_1 \mapsto m_{c'1} \\ w_2 \mapsto m_{c2} \\ w_3 \mapsto m_{c'3} \end{array}, \begin{array}{l} w_1 \mapsto m_{c'1} \\ w_2 \mapsto m'_{c2} \\ w_3 \mapsto m_{c'3} \end{array}, \begin{array}{l} w_1 \mapsto m'_{c'1} \\ w_2 \mapsto m_{c2} \\ w_3 \mapsto m'_{c'3} \end{array}, \begin{array}{l} w_1 \mapsto m'_{c'1} \\ w_2 \mapsto m'_{c2} \\ w_3 \mapsto m'_{c'3} \end{array}\right\}$$

Refinement relation (low-level)

- Input We are interested in the distinguishing capacities of the adversary on inputs that directly correspond to stores. These are called store projections and are random variables $(\mathrm{Lay} \to \mathrm{Mem}_{\perp,\xi})$ of the form $w \mapsto w(s)$ for some store s (but also allow $\perp$ and ξ).
- Output For $X, Y \in \mathscr{H}(\mathrm{Lay} \to \mathrm{Mem}_{\perp,\xi})$, $X \leq_L Y$ iff (roughly) the public parts of the random variables in X are less than those of Y *with probability greater than* δ. More precisely, for all $\zeta \in X$, there exists $\zeta' \in Y$ such that:
  - $\zeta_L \leq \zeta'_L$, or
  - $P(\zeta(w) \in \{\xi, \perp\}) \geq \delta$ and $P(\zeta'(w) = \xi) \geq \delta$.
- Command refinement c refines c', written $c \leq_L c'$, when:

$$\forall s \in \text{stores}.\ [\![c]\!](w \mapsto w(s)) \leq_L [\![c']\!](w \mapsto w(s))$$

Simulation relation

- We mostly consider public contexts containing safe commands.
- Safe commands transform store projections into store projection sets, which are (downsets of) sets of
  - store projections, or
  - random variables with high probability of error ($> \delta$).
- We project those semantics and define a simulation relation similar to the high-level one.

Theorem 2

For all safe low-level commands c, c′, we have:

$$[\![c]\!] \preceq [\![c']\!] \iff \forall \text{ low-level public } C[\,].\ [\![C[c]]\!] \leq_L [\![C[c']]\!]$$

Simulation relation - more detail

- Begin with $R \subseteq \mathscr{H}(\text{private-stores}_{\perp,\xi})^2$, subject to conditions
- Then construct $R^+ \subseteq \mathscr{H}(\text{stores}_{\perp,\xi})^2$ from R, much as before
- Then construct $R^\times \subseteq \mathscr{H}(\text{Lay} \to \text{Mem}_{\xi,\perp})^2$ from R^+ by:

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

(where χ extracts the stores, bottom, and, possibly, error from a store projection set).

- Then

$$[\![c]\!] \preceq [\![c']\!] \ \equiv_{\text{def}}\ \forall X, Y.\, X R^\times Y \Rightarrow [\![c]\!](X) R^\times [\![c']\!](Y)$$

Compilation

High-level commands c are compiled to low-level commands $c^{\downarrow}$ by:

$$\begin{array}{lcl} (!l_{\mathtt{loc}})^{\downarrow} & = & !l_{\mathtt{nat}} \\ (l_{\mathtt{loc}} := e)^{\downarrow} & = & l_{\mathtt{nat}} := e^{\downarrow} \end{array}$$

Theorem 3

For all high-level commands c, c' we have: $c \preceq c'$ iff $c^{\downarrow} \preceq c'^{\downarrow}$

Theorem 4

For all high-level commands c, c',

$$\forall \textit{high-level public } C[\,].\, [\![C[c]]\!] \leq_L [\![C[c']]\!]$$

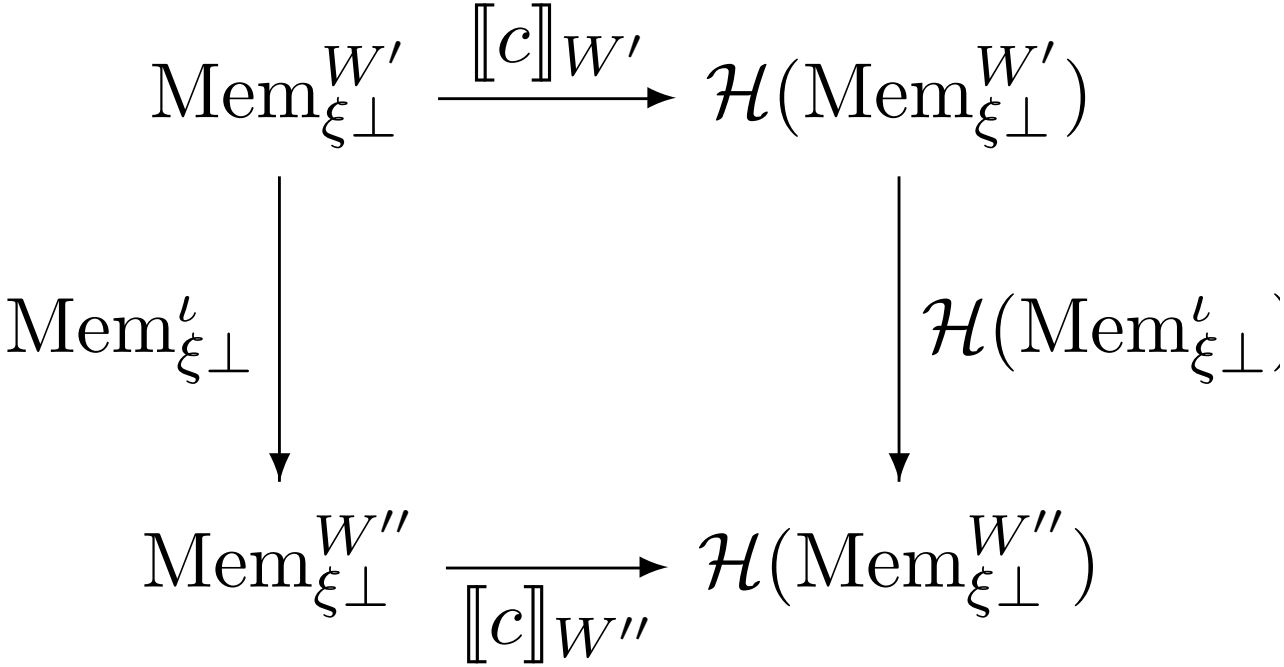
holds iff

$$\forall \textit{low-level public } C[\,].\, [\![C[c^{\downarrow}]]\!] \leq_L [\![C[c'^{\downarrow}]]\!]$$

does

Naturality of the semantics

Parameterising over sets of layouts:



Suggests working with presheaves.

Summary of this work

- A semantic approach to layout randomization

$$\begin{array}{ccc} [\![c]\!] \preceq [\![c']\!] & \Leftrightarrow & \forall \text{ public } C[\,].\, [\![C[c]]\!] \leq_L [\![C[c']]\!] \\ \Updownarrow & & \\ [\![c^{\downarrow}]\!] \preceq [\![c'^{\downarrow}]\!] & \Leftrightarrow & \forall \text{ public } C[\,].\, [\![C[c^{\downarrow}]]\!] \leq_L [\![C[c'^{\downarrow}]]\!] \end{array}$$

- Nondeterminism (and its interaction with probabilities).
- Concurrency would be an interesting next step (or higher-order)

Summary of this work

- A semantic approach to layout randomization

$$\begin{array}{ccc} [\![c]\!] \preceq [\![c']\!] & \Leftrightarrow & \forall \text{ public } C[\,].\, [\![C[c]]\!] \leq_L [\![C[c']]\!] \\ \Updownarrow & & \\ [\![c^{\downarrow}]\!] \preceq [\![c'^{\downarrow}]\!] & \Leftrightarrow & \forall \text{ public } C[\,].\, [\![C[c^{\downarrow}]]\!] \leq_L [\![C[c'^{\downarrow}]]\!] \end{array}$$

- Nondeterminism (and its interaction with probabilities).
- Concurrency would be an interesting next step (or higher-order)
- But there is still something to understand.

What is the corresponding operational semantics?

- To make sense of our semantics, we want operational/denotational consistency, akin to:

$$\zeta' \in [\![c]\!](\zeta) \iff \forall w.\, w \models \langle c, \zeta(w)\rangle \rightarrow^* \zeta'(w)$$

- Once again, the nondeterminism is chosen after the layout.
- We set the nondeterminism with an oracle, then the operational semantics is deterministic:

$$\zeta' \in [\![c]\!](\zeta) \iff \exists \pi. \forall w.\, w \models \langle c, \zeta(w), \pi\rangle \rightarrow^* \zeta'(w)$$