Verifying Probabilistic Correctness in Isabelle with pGCL

David Cock

30 November 2012

NICTA

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council
NICTA Funding and Supporting Members and Partners

Outline

- Stochastic Behaviour in Systems
- Functional vs. Probabilistic Verification
- pGCL in Isabelle/HOL
- Example: Lattice-Lottery Scheduler

Sources of Uncertainty

We like certainty.

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sometimes however, we're forced to live with uncertainty.

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sometimes however, we're forced to live with uncertainty.

Some things are inherently unpredictable:

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sometimes however, we're forced to live with uncertainty.

Some things are inherently unpredictable:

- Device failure.

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sometimes however, we're forced to live with uncertainty.

Some things are inherently unpredictable:

Device failure.

Some things are simply too complex to model:

Sources of Uncertainty

We like certainty.

The L4.verified proof tells us that if its assumptions are satisfied, seL4 will *definitely* not crash.

Sometimes however, we're forced to live with uncertainty.

Some things are inherently unpredictable:

- Device failure.

Some things are simply too complex to model:

- A modern processor.

Probability Refines Nondeterminism

Classical nondeterminism is the ultimate in pessimism: Anything that *can* happen *will* happen.

If we know how events are distributed, we can do better.

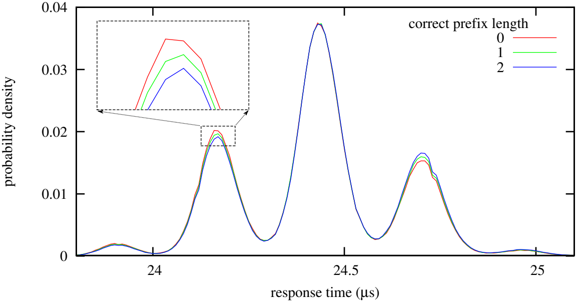
Probabilistic models are a halfway-house between full nondeterminism and full predictability.

Probabilistic guarantees are relevant both for security, and for reliability.

Our current work is on probabilistic security guarantees.

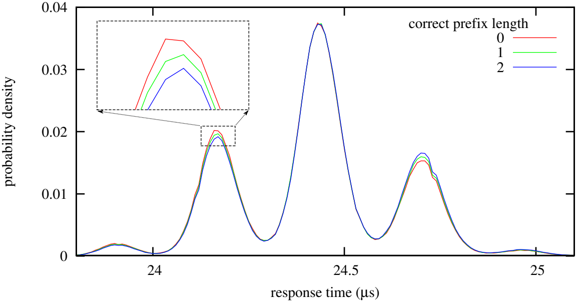
Why is this relevant in systems?

Feed a secret string and a guess to strcmp:



Why is this relevant in systems?

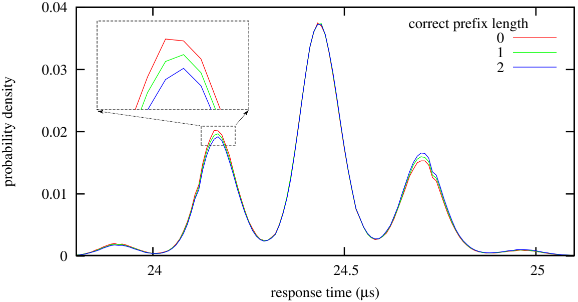
Feed a secret string and a guess to strcmp:



This is a side-channel, which exposes the secret.

Why is this relevant in systems?

Feed a secret string and a guess to strcmp:



This is a side-channel, which exposes the secret.

How bad is it? How can we mitigate it?
How will it behave in a larger system?

Proving Security

Probabilistic verification can help us answer these questions.

Proving Security

Probabilistic verification can help us answer these questions.

We want to show something like:

$$\wp\ ((r, \tau) := \mathtt{strcmp}(g, s);$$
$$g := \mathtt{cleverness}(r, \tau, g))\ (g = s) \leq 2^{-100}$$

Proving Security

Probabilistic verification can help us answer these questions.

We want to show something like:

$$\wp\ ((r, \tau) := \texttt{strcmp}(g, s);\\ g := \texttt{cleverness}(r, \tau, g))\ (g = s) \leq 2^{-100}$$

Formulating this rigorously is the subject of our existing work.

Mechanising this work in Isabelle/HOL ensures our reasoning is sound, and scalable to large problems.

We use pGCL, an extension of Dijkstra's GCL with probability.

Outline

- Stochastic Behaviour in Systems
- Functional vs. Probabilistic Verification
- pGCL in Isabelle/HOL
- Example: Lattice-Lottery Scheduler

Judgements on Programs

How do we interpret this?

$$\{x = 0\}\ y := x^2 \{y = x\}$$

Judgements on Programs

How do we interpret this?

$$\{x = 0\}\ y := x^2 \{y = x\}$$

This relates a program to an annotation.
If $x = 0$ holds before, then $y = x$ holds afterwards.

Is $x = 0$ maximal?

Judgements on Programs

How do we interpret this?

$$\{x = 0\}\ y := x^2 \{y = x\}$$

This relates a program to an annotation.
If $x = 0$ holds before, then $y = x$ holds afterwards.

Is $x = 0$ maximal? No, $x = 1$ works too.

Judgements on Programs

How do we interpret this?

$$\{x = 0\}\ y := x^2 \{y = x\}$$

This relates a program to an annotation.
If $x = 0$ holds before, then $y = x$ holds afterwards.

Is $x = 0$ maximal? No, $x = 1$ works too.

$\{x = 0 \vee x = 1\}$ *is* maximal,
it is the *weakest precondition* of $\{y = x\}$.

Judgements on Programs

How do we interpret this?

$$\{x = 0\}\ y := x^2 \{y = x\}$$

This relates a program to an annotation.
If $x = 0$ holds before, then $y = x$ holds afterwards.

Is $x = 0$ maximal? No, $x = 1$ works too.

$\{x = 0 \vee x = 1\}$ *is* maximal,
it is the *weakest precondition* of $\{y = x\}$.

$$\wp\ a\ Q \equiv \sup\,\{P | P\ a\ Q\}$$

$$\{R\} \leq \{S\} \equiv R \vdash S \equiv \forall s.\ R\ s \rightarrow S\ s$$

Nondeterminism

Nondeterminism allows us to underspecify a program.

Nondeterminism

Nondeterminism allows us to underspecify a program.

We write $a \sqcap b$ for 'Do either *a* or *b*'.

We let a demon make the choice, who tries to trip us up.

Nondeterminism

Nondeterminism allows us to underspecify a program.

We write $a \sqcap b$ for 'Do either a or b'.

We let a demon make the choice, who tries to trip us up.

What is $\wp\,(y := x^2 \sqcap y := 2x)\,(y = x)$?

Nondeterminism

Nondeterminism allows us to underspecify a program.

We write $a \sqcap b$ for 'Do either a or b'.

We let a demon make the choice, who tries to trip us up.

What is $\wp\ (y := x^2 \sqcap y := 2x)\ (y = x)$?

Algebraically: $\wp\ (a \sqcap b)\ Q = \wp\ a\ Q \cap \wp\ b\ Q$

Nondeterminism

Nondeterminism allows us to underspecify a program.

We write $a \sqcap b$ for 'Do either a or b'.

We let a demon make the choice, who tries to trip us up.

What is $\wp\ (y := x^2 \sqcap y := 2x)\ (y = x)$?

Algebraically: $\wp\ (a \sqcap b)\ Q = \wp\ a\ Q \cap \wp\ b\ Q$

Thus $P = \{x = 0 \vee x = 1\} \cap \{x = 0\} = \{x = 0\}$.

We are treating annotations as sets.

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector:

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector: $«P»\ s \equiv 1$ if $s \in P$ else 0.

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector: $«P»\ s \equiv 1$ if $s \in P$ else 0.

We can still order these: $«P» \leq «Q» \equiv \forall s. «P»\ s \leq «Q»\ s$

Note: $\wp\ (a \sqcap b)\ «Q» = \min\ (\wp\ a\ «Q»)\ (\wp\ b\ «Q»)$.

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector: $«P»\ s \equiv 1$ if $s \in P$ else 0.

We can still order these: $«P» \leq «Q» \equiv \forall s. «P»\ s \leq «Q»\ s$

Note: $\wp\ (a \sqcap b)\ «Q» = \min\ (\wp\ a\ «Q»)\ (\wp\ b\ «Q»)$.

The 'weakest precondition' is the *least* value that the postcondition may take, from a given initial state.

Quantitative Predicates

NICTA

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector: $«P»\ s \equiv 1$ if $s \in P$ else 0.

We can still order these: $«P» \leq «Q» \equiv \forall s. «P»\ s \leq «Q»\ s$

Note: $\wp\ (a \sqcap b)\ «Q» = \min\ (\wp\ a\ «Q»)\ (\wp\ b\ «Q»)$.

The ‘weakest precondition’ is the *least* value that the postcondition may take, from a given initial state.

It is the pessimistic *expected value* of the postcondition.

Quantitative Predicates

So far, $\wp$ defines a set; What about $\wp$ as a probability?

Identify a set with its selector: $«P»\ s \equiv 1$ if $s \in P$ else 0.

We can still order these: $«P» \leq «Q» \equiv \forall s. «P»\ s \leq «Q»\ s$

Note: $\wp\ (a \sqcap b)\ «Q» = \min\ (\wp\ a\ «Q»)\ (\wp\ b\ «Q»)$.

The 'weakest precondition' is the *least* value that the postcondition may take, from a given initial state.

It is the pessimistic *expected value* of the postcondition.

These quantitative predicates are called *expectations*.

Probabilistic Choice

What if the demon were a gambler?

Probabilistic Choice

What if the demon were a gambler?

$a \ {}_{1/2}{\oplus} \ b$ means 'flip a coin — if heads a otherwise b'.

What should $\wp \ (y := x^2 \ {}_{1/2}{\oplus} \ y := 2x) \ (y = x)$ be?

Probabilistic Choice

What if the demon were a gambler?

$a\ {}_{1/2}\oplus\ b$ means 'flip a coin — if heads a otherwise b'.

What should $\wp\ (y := x^2\ {}_{1/2}\oplus\ y := 2x)\ (y = x)$ be?

For an expectation, we'd take the weighted average:

$$\wp\ (a\ {}_{p}\oplus\ b)\ F = p \times \wp\ a\ F + (1 - p) \times \wp\ b\ F$$

Probabilistic Choice

What if the demon were a gambler?

$a \;{}_{1/2}\!\oplus\; b$ means 'flip a coin — if heads *a* otherwise *b*'.

What should $\wp\,(y := x^2 \;{}_{1/2}\!\oplus\; y := 2x)\;(y = x)$ be?

For an expectation, we'd take the weighted average:

$$\wp\,(a \;{}_{p}\!\oplus\; b)\;F = p \times \wp\;a\;F + (1 - p) \times \wp\;b\;F$$

$\wp\,(a \;{}_{p}\!\oplus\; b)\;(y = x)\;s$ is the *probability* that, if we start in state s, $y = x$ holds in the final state.

Probabilistic Choice

What if the demon were a gambler?

$a \ {}_{1/2}{\oplus}\ b$ means 'flip a coin — if heads a otherwise b'.

What should $\wp\ (y := x^2\ {}_{1/2}{\oplus}\ y := 2x)\ (y = x)$ be?

For an expectation, we'd take the weighted average:

$$\wp\ (a\ {}_{p}{\oplus}\ b)\ F = p \times \wp\ a\ F + (1 - p) \times \wp\ b\ F$$

$\wp\ (a\ {}_{p}{\oplus}\ b)\ (y = x)\ s$ is the *probability* that, if we start in state s, $y = x$ holds in the final state.

$\wp\ (a\ {}_{p}{\oplus}\ b)\ (y = x)\ 0 = 1$ and $\wp\ (a\ {}_{p}{\oplus}\ b)\ (y = x)\ 1 = 1/2$.

All other values are zero.

Combining Probability and Nondeterminism

How about this?

$$E = \wp\left((y := x^2 \ {}_{1/2}\!\oplus\ y := 2x) \sqcap \\ (y := x^2 \ {}_{1/3}\!\oplus\ y := 2x)\right)(y = x)$$

Combining Probability and Nondeterminism

How about this?

$$E = \wp \left((y := x^2 \;{}_{1/2}\!\oplus\; y := 2x) \sqcap \\ (y := x^2 \;{}_{1/3}\!\oplus\; y := 2x) \right) (y = x)$$

Simply apply both rules:

$$E\ x = \min\ (1/2 \times «x = 0 \lor x = 1» + 1/2 \times «x = 0») \\ (1/3 \times «x = 0 \lor x = 1» + 2/3 \times «x = 0»)$$

This time, $E\ 0 = 1$ and $E\ 1 = 1/3$.

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Combining Probability and Nondeterminism

How about this?

$$E = \wp \Big((y := x^2 \ {}_{1/2}\!\oplus \ y := 2x) \sqcap \\ (y := x^2 \ {}_{1/3}\!\oplus \ y := 2x) \Big) \ (y = x)$$

Simply apply both rules:

$$E \ x = \min \ (1/2 \times «x = 0 \lor x = 1» + 1/2 \times «x = 0») \\ (1/3 \times «x = 0 \lor x = 1» + 2/3 \times «x = 0»)$$

This time, $E\ 0 = 1$ and $E\ 1 = 1/3$.

$E\ x$ is the *minimum* probability that $y = x$ will hold.

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

pGCL

These are basics of pGCL (Morgan & McIver, 2004).

It's a formal model of computation incorporating probability and nondeterminism.

In the remainder of the talk I will introduce our mechanisation in Isabelle/HOL, and our work on the probabilistic verification of systems software.

Outline

- Stochastic Behaviour in Systems
- Functional vs. Probabilistic Verification
- pGCL in Isabelle/HOL
- Example: Lattice-Lottery Scheduler

Expectations

The pGCL package provides a shallow embedding into HOL.

Expectations

The pGCL package provides a shallow embedding into HOL.

Expectations use the standard real number type:

$$E :: \sigma \Rightarrow \mathbb{R}$$

This allows us to use existing results directly.

Expectations

The pGCL package provides a shallow embedding into HOL.

Expectations use the standard real number type:

$$E :: \sigma \Rightarrow \mathbb{R}$$

This allows us to use existing results directly.

Expectations are nonnegative and bounded:

$$\text{nneg } E \equiv \forall s.\ 0 \leq E\ s \qquad \text{bounded } E \equiv \exists b.\ \forall s.\ E\ s \leq b$$

Expectations

The pGCL package provides a shallow embedding into HOL.

Expectations use the standard real number type:

$$E :: \sigma \Rightarrow \mathbb{R}$$

This allows us to use existing results directly.

Expectations are nonnegative and bounded:

$$\text{nneg } E \equiv \forall s.\ 0 \leq E\ s \qquad \text{bounded } E \equiv \exists b.\ \forall s.\ E\ s \leq b$$

The state space need not, in general, be finite.

Expectation Transformers

Programs are expectation transformers:

$$\wp\, a :: (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

Expectation Transformers

Programs are expectation transformers:

$$\wp\, a :: (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

We usually restrict our attention to *healthy* transformers:

Expectation Transformers

Programs are expectation transformers:

$$\wp\, a :: (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

We usually restrict our attention to *healthy* transformers:

$$\begin{aligned}&\forall P\, b.\ \text{bounded\_by}\ b\ P \wedge \text{nneg}\ P \rightarrow \\ &\qquad \text{bounded\_by}\ b\ (t\ P) \wedge \text{nneg}\ (t\ P)\end{aligned}$$

Expectation Transformers

Programs are expectation transformers:

$$\wp\, a :: (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

We usually restrict our attention to *healthy* transformers:

$$\forall P\, b.\ \text{bounded\_by}\ b\ P \wedge \text{nneg}\ P \rightarrow$$
$$\quad \text{bounded\_by}\ b\ (t\ P) \wedge \text{nneg}\ (t\ P)$$
$$\forall P\, Q.\ (\text{sound}\ P \wedge \text{sound}\ Q \wedge P \vdash Q) \longrightarrow (t\ P) \vdash (t\ Q)$$

Expectation Transformers

Programs are expectation transformers:

$$\wp\, a :: (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

We usually restrict our attention to *healthy* transformers:

$$\forall P\, b.\ \text{bounded\_by}\ b\ P \wedge \text{nneg}\ P \rightarrow$$
$$\qquad \text{bounded\_by}\ b\ (t\ P) \wedge \text{nneg}\ (t\ P)$$
$$\forall P\, Q.\ (\text{sound}\ P \wedge \text{sound}\ Q \wedge P \vdash Q) \longrightarrow (t\ P) \vdash (t\ Q)$$
$$\forall P\, c\, s.\ (\text{sound}\ P \wedge 0 < c) \longrightarrow c \times t\ P\ s = t\ (\lambda s.\ c \times P\ s)\ s$$

A few primitives

$$\text{Abort} \equiv \lambda ab\, P.\ \text{if } ab \text{ then } \lambda s.\ 0 \text{ else } \lambda s.\ \text{bound\_of } P$$

We model both strict (WP) and liberal (WLP) semantics.

All these primitives are healthy.

A few primitives

$$\text{Abort} \equiv \lambda ab\, P.\ \text{if } ab \text{ then } \lambda s.\ 0 \text{ else } \lambda s.\ \text{bound\_of } P$$

$$a \sqcap b \equiv \lambda ab\, P\, s.\ \min\ (a\ ab\ P\ s)\ (b\ ab\ P\ s)$$

We model both strict (WP) and liberal (WLP) semantics.

All these primitives are healthy.

A few primitives

$$\text{Abort} \equiv \lambda ab\, P.\ \text{if } ab \text{ then } \lambda s.\ 0 \text{ else } \lambda s.\ \text{bound\_of } P$$

$$a \sqcap b \equiv \lambda ab\, P\, s.\ \ \min\ (a\ ab\ P\ s)\ (b\ ab\ P\ s)$$

$$a\ {}_p\!\oplus\ b \equiv \lambda ab\, P\, s.\ p \times (a\ ab\ P\ s) + (1 - p) \times (b\ ab\ P\ s)$$

We model both strict (WP) and liberal (WLP) semantics.

All these primitives are healthy.

A few primitives

$$\text{Abort} \equiv \lambda ab\, P.\ \text{if } ab \text{ then } \lambda s.\ 0 \text{ else } \lambda s.\ \text{bound\_of } P$$

$$a \sqcap b \equiv \lambda ab\, P\, s.\ \min\ (a\ ab\ P\ s)\ (b\ ab\ P\ s)$$

$$a\ {}_p\!\oplus\ b \equiv \lambda ab\, P\, s.\ p \times (a\ ab\ P\ s) + (1 - p) \times (b\ ab\ P\ s)$$

$$\wp\ a \equiv a\ \text{True}$$

We model both strict (WP) and liberal (WLP) semantics.

All these primitives are healthy.

Embedding a Monad

The shallow embedding makes it easy to embed the L4.verified nondeterministic monad:

$$\text{Exec} :: (\sigma \Rightarrow (\alpha \times \sigma)\ \text{set}) \Rightarrow \text{bool} \Rightarrow (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

$$\text{Exec}\ M \equiv \lambda ab\ R\ s.\ \text{glb}\ \{R\ (\text{snd}\ sa).\ sa \in M\ s\}$$

Embedding a Monad

The shallow embedding makes it easy to embed the L4.verified nondeterministic monad:

$$\text{Exec} :: (\sigma \Rightarrow (\alpha \times \sigma)\ \text{set}) \Rightarrow \text{bool} \Rightarrow (\sigma \Rightarrow \mathbb{R}) \Rightarrow \sigma \Rightarrow \mathbb{R}$$

$$\text{Exec}\ M \equiv \lambda ab\ R\ s.\ \text{glb}\ \{R\ (\text{snd}\ sa).\ sa \in M\ s\}$$

We lift Hoare triples to probabilistic entailments:

$$\frac{\{P\}\ prog\ \{\lambda r\ s.\ Q\ s\} \quad \forall s.\ prog\ s \neq \{\} \quad \exists s.\ P\ s}{«P» \vdash \wp\ prog\ «Q»}\ \textsc{wp\_Exec}$$

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Outline

- Stochastic Behaviour in Systems
- Functional vs. Probabilistic Verification
- pGCL in Isabelle/HOL
- Example: Lattice-Lottery Scheduler

Refinement

One of the principle tools in verification is *refinement*.

Refinement

One of the principle tools in verification is *refinement*.

A refinement relation allows us to transfer properties from *specification* to *implementation*:

$$\frac{a \sqsubseteq b \quad E \vdash \wp.a.F}{E \vdash \wp.b.F}$$

Refinement

One of the principle tools in verification is *refinement*.

A refinement relation allows us to transfer properties from *specification* to *implementation*:

$$\frac{a \sqsubseteq b \quad E \vdash \wp.a.F}{E \vdash \wp.b.F}$$

Given E, if a establishes F, then so does b or:

$$\wp.a.F \leq \wp.b.F$$

Refinement

One of the principle tools in verification is *refinement*.

A refinement relation allows us to transfer properties from *specification* to *implementation*:

$$\frac{a \sqsubseteq b \quad E \vdash \wp.a.F}{E \vdash \wp.b.F}$$

Given E, if a establishes F, then so does b or:

$$\wp.a.F \leq \wp.b.F$$

In pGCL, an implementation establishes any property with at least as great a probability as its specification.

Lattice Scheduling

An approach to efficiently eliminating leaks through shared state e.g. caches.

Only switch to a domain with higher clearance, or to the downgrader, which clears the cache:

Lattice Scheduling

An approach to efficiently eliminating leaks through shared state e.g. caches.

Only switch to a domain with higher clearance, or to the downgrader, which clears the cache:

$$\text{scheduleL} \equiv cd :\in \lambda s.\ \{n | (cd, n) \in S\}$$

Lattice Scheduling

An approach to efficiently eliminating leaks through shared state e.g. caches.

Only switch to a domain with higher clearance, or to the downgrader, which clears the cache:

$$\text{scheduleL} \equiv cd :\in \lambda s.\ \{n | (cd, n) \in S\}$$

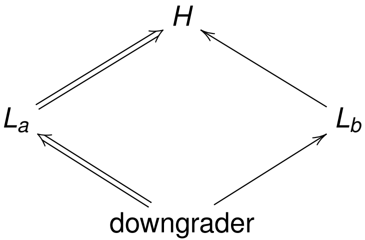
The security property:

$$\forall c, n.\ (c, n) \in S \rightarrow \text{sec\_class}.c \leq \text{sec\_class}.n \vee \\ n = \text{downgrader}$$

Unfairness

$$\begin{tikzcd}
 & H & \\
L_a \arrow[ur] & & L_b \arrow[ul] \\
 & \text{downgrader} \arrow[ul] \arrow[ur] &
\end{tikzcd}$$

Unfairness



Unfairness

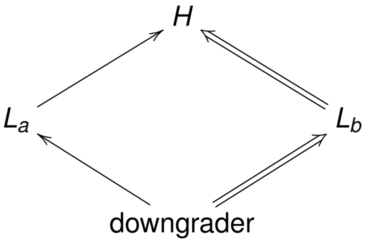
$$\begin{tikzcd} & H & \\ L_a \arrow[ur] & & L_b \arrow[ul, Rightarrow] \\ & \text{downgrader} \arrow[ul] \arrow[ur, Rightarrow] & \end{tikzcd}$$

Unfairness

$$\begin{tikzcd}
& H & \\
L_a \arrow[ur] & & L_b \arrow[ul, Rightarrow] \\
& \text{downgrader} \arrow[ul] \arrow[ur, Rightarrow] &
\end{tikzcd}$$

A single-period schedule cannot include both L_a and L_b.

Unfairness



A single-period schedule cannot include both L_a and L_b.

A nondeterministic scheduler might simply always pick L_b.

Stochastic Behaviour in Systems
Functional vs. Probabilistic Verification
pGCL in Isabelle/HOL
Example: Lattice-Lottery Scheduler

Randomised Lattice Scheduling

We'd still like to have asymptotic fairness between domains.

Randomised Lattice Scheduling

We'd still like to have asymptotic fairness between domains.

Start by randomising:

$$\text{scheduleR} \equiv cd :\in \text{UNIV at} (\lambda s\, n.\ T\ (cd, n))$$

Randomised Lattice Scheduling

We'd still like to have asymptotic fairness between domains.

Start by randomising:

$$\text{scheduleR} \equiv cd :\in \text{UNIV at } (\lambda s\, n.\ T\ (cd, n))$$

If the matrix T satisfies:

$$\forall c\, n.\ 0 < T\ (c, n) \to (c, n) \in S$$

we have refinement, scheduleL $\sqsubseteq$ scheduleR.

Randomised Lattice Scheduling

We'd still like to have asymptotic fairness between domains.

Start by randomising:

$$\text{scheduleR} \equiv cd :\in \text{UNIV at } (\lambda s\, n.\ T\ (cd, n))$$

If the matrix T satisfies:

$$\forall c\, n.\ 0 < T\ (c, n) \rightarrow (c, n) \in S$$

we have refinement, scheduleL $\sqsubseteq$ scheduleR.

This scheduler is a Markov process, and if T is irreducible and positive recurrent, there exists a stationary distribution.

Lottery Scheduling

NICTA

An efficient implementation might use a lottery:

$$
\begin{aligned}
&\text{scheduleM } t \equiv \text{do} \\
&\quad c \leftarrow \text{gets cd};\ \ l \leftarrow \text{gets lottery}; \\
&\quad \text{let } n = l\ c\ t \text{ in modify}(\lambda s.\ s(\text{cd} := n)) \\
&\text{od}
\end{aligned}
$$

The lottery has type: domain $\Rightarrow$ word32 $\Rightarrow$ domain.

Lottery Scheduling

An efficient implementation might use a lottery:

$$\begin{aligned}
&\text{scheduleM } t \equiv \text{do} \\
&\quad c \leftarrow \text{gets cd};\ \ l \leftarrow \text{gets lottery}; \\
&\quad \text{let } n = l\ c\ t \text{ in modify}(\lambda s.\ s(\text{cd} := n)) \\
&\text{od}
\end{aligned}$$

The lottery has type: domain $\Rightarrow$ word32 $\Rightarrow$ domain.

We chain in probability from above:

$$\text{scheduleC} \equiv t \text{ from UNIV at } 2^{-32} \text{ in Exec (scheduleM } t)$$

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Data Refinement

We cannot show that scheduleR $\sqsubseteq$ scheduleC, as they operate on different state spaces:

$$\textbf{record}\ \text{stateA} = \mathit{cd} :: \text{domain}$$

$$\begin{aligned}\textbf{record}\ \text{stateC} = \mathit{cd} &:: \text{domain},\\ \text{lottery} &:: \text{domain} \Rightarrow \text{word32} \Rightarrow \text{domain}\end{aligned}$$

Data Refinement

We cannot show that scheduleR $\sqsubseteq$ scheduleC, as they operate on different state spaces:

$$\begin{aligned}
&\textbf{record}\ \text{stateA} = \mathit{cd} :: \text{domain} \\
&\textbf{record}\ \text{stateC} = \mathit{cd} :: \text{domain}, \\
&\qquad\qquad\qquad \text{lottery} :: \text{domain} \Rightarrow \text{word32} \Rightarrow \text{domain}
\end{aligned}$$

The lottery is an implementation detail, only *cd* matters.

Data Refinement

We cannot show that scheduleR $\sqsubseteq$ scheduleC, as they operate on different state spaces:

$$\begin{aligned}
&\textbf{record}\ \text{stateA} = \mathit{cd} :: \text{domain} \\
&\textbf{record}\ \text{stateC} = \mathit{cd} :: \text{domain}, \\
&\qquad\qquad\qquad\quad \text{lottery} :: \text{domain} \Rightarrow \text{word32} \Rightarrow \text{domain}
\end{aligned}$$

The lottery is an implementation detail, only *cd* matters.

Take the natural projection: $\phi :: \text{stateC} \Rightarrow \text{stateA}$.

Data Refinement

We define *data refinement*, $\sqsubseteq_{\phi,Pre}$:

$$\frac{a \sqsubseteq_{\phi,Pre} b \quad E \vdash \wp\, a\, F \quad Pre\ s}{(E \circ \phi)\ s \vdash \wp\, b\, (F \circ \phi)\ s}$$

NICTA

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Data Refinement

We define *data refinement*, $\sqsubseteq_{\phi,Pre}$:

$$\frac{a \sqsubseteq_{\phi,Pre} b \quad E \vdash \wp\, a\, F \quad Pre\ s}{(E \circ \phi)\ s \vdash \wp\, b\, (F \circ \phi)\ s}$$

If the ticket distribution represents the transition matrix:

$$LR\ s \equiv \forall c, n.\ T(c, n) = \sum_{t.\ \text{lottery}\ s\ c\ t = n} 2^{-32}$$

we have another refinement step:

Data Refinement

We define *data refinement*, $\sqsubseteq_{\phi,Pre}$:

$$\frac{a \sqsubseteq_{\phi,Pre} b \quad E \vdash \wp\, a\, F \quad Pre\ s}{(E \circ \phi)\ s \vdash \wp\, b\, (F \circ \phi)\ s}$$

If the ticket distribution represents the transition matrix:

$$LR\ s \equiv \forall c, n.\ T(c, n) = \sum_{t.\ \text{lottery}\ s\ c\ t=n} 2^{-32}$$

we have another refinement step:

$$\text{scheduleL} \sqsubseteq \text{scheduleR}$$

Data Refinement

We define *data refinement*, $\sqsubseteq_{\phi,Pre}$:

$$\frac{a \sqsubseteq_{\phi,Pre} b \quad E \vdash \wp\ a\ F \quad Pre\ s}{(E \circ \phi)\ s \vdash \wp\ b\ (F \circ \phi)\ s}$$

If the ticket distribution represents the transition matrix:

$$LR\ s \equiv \forall c, n.\ T(c,n) = \sum_{t.\ \text{lottery}\ s\ c\ t=n} 2^{-32}$$

we have another refinement step:

$$\text{scheduleL} \sqsubseteq \text{scheduleR} \sqsubseteq_{\phi,LR} \text{scheduleC}$$

NICTA

Stochastic Behaviour in Systems

Functional vs. Probabilistic Verification

pGCL in Isabelle/HOL

Example: Lattice-Lottery Scheduler

Attaching the Kernel

Finally, we attach a kernel model:

$$\text{stepKernel} \equiv \text{callKernel}; \text{scheduleC}$$

Attaching the Kernel

Finally, we attach a kernel model:

$$\text{stepKernel} \equiv \text{callKernel}; \text{scheduleC}$$

We need only a few high-level properties, including:

$$\{cd = d\}\ \text{callKernel}\ \{cd = d\}$$

which is a specification in the L4.verified Hoare logic, from which we establish:

Attaching the Kernel

Finally, we attach a kernel model:

$$\text{stepKernel} \equiv \text{callKernel}; \text{scheduleC}$$

We need only a few high-level properties, including:

$$\{cd = d\}\ \text{callKernel}\ \{cd = d\}$$

which is a specification in the L4.verified Hoare logic, from which we establish:

$$\text{Skip} \sqsubseteq_{\phi,LR} \text{callKernel}$$

Attaching the Kernel

Finally, we attach a kernel model:

$$\text{stepKernel} \equiv \text{callKernel}; \text{scheduleC}$$

We need only a few high-level properties, including:

$$\{cd = d\}\ \text{callKernel}\ \{cd = d\}$$

which is a specification in the L4.verified Hoare logic, from which we establish:

$$\text{Skip} \sqsubseteq_{\phi,LR} \text{callKernel}$$

The kernel may modify the lottery!

Attaching the Kernel

If the kernel additionally preserves the lottery relation:

$$\{LR\}\ \text{callKernel}\ \{LR\}$$

then we have the full refinement chain:

Attaching the Kernel

If the kernel additionally preserves the lottery relation:

$$\{LR\}\ \text{callKernel}\ \{LR\}$$

then we have the full refinement chain:

$$\text{scheduleL} \sqsubseteq \text{scheduleR}$$

Attaching the Kernel

If the kernel additionally preserves the lottery relation:

$$\{LR\}\ \text{callKernel}\ \{LR\}$$

then we have the full refinement chain:

$$\text{scheduleL} \sqsubseteq \text{scheduleR} \sqsubseteq_{\phi,LR} \text{stepKernel}$$

Attaching the Kernel

If the kernel additionally preserves the lottery relation:

$$\{LR\}\ \text{callKernel}\ \{LR\}$$

then we have the full refinement chain:

$$\text{scheduleL} \sqsubseteq \text{scheduleR} \sqsubseteq_{\phi,LR} \text{stepKernel}$$

The kernel implements a fair, secure scheduler.

Summary

We have:

Summary

We have:

- Motivated probabilistic verification for systems.

Summary

We have:

- Motivated probabilistic verification for systems.
- Mechanised pGCL in Isabelle/HOL.

Summary

We have:

- Motivated probabilistic verification for systems.
- Mechanised pGCL in Isabelle/HOL.
- Verified a randomised scheduler.

Questions?