From Operational Models to Information Theory
Side Channels in pGCL with Isabelle

David Cock

14 July 2014

Australian Government
Department of Broadband, Communications and the Digital Economy
Australian Research Council

NICTA Funding and Supporting Members and Partners

Australian National University
UNSW THE UNIVERSITY OF NEW SOUTH WALES
NSW GOVERNMENT Trade & Investment
THE UNIVERSITY OF SYDNEY
Queensland Government

Overview

A pen-and-paper proof formalised in Isabelle.

- Linking operational semantics and information theory.
- Testing the limits of pGCL in Isabelle.
- Hand calculation eliminated.
- Composes with L4.verified stack.

Overview

A pen-and-paper proof formalised in Isabelle.

- Linking operational semantics and information theory.
- Testing the limits of pGCL in Isabelle.
- Hand calculation eliminated.
- Composes with L4.verified stack.

What's in the talk?

Overview

A pen-and-paper proof formalised in Isabelle.

- Linking operational semantics and information theory.
- Testing the limits of pGCL in Isabelle.
- Hand calculation eliminated.
- Composes with L4.verified stack.

What's in the talk?

- What was the original result, and why formalise it?

Overview

NICTA

A pen-and-paper proof formalised in Isabelle.

- Linking operational semantics and information theory.
- Testing the limits of pGCL in Isabelle.
- Hand calculation eliminated.
- Composes with L4.verified stack.

What's in the talk?

- What was the original result, and why formalise it?
- How was it formalised?

Overview

NICTA

A pen-and-paper proof formalised in Isabelle.

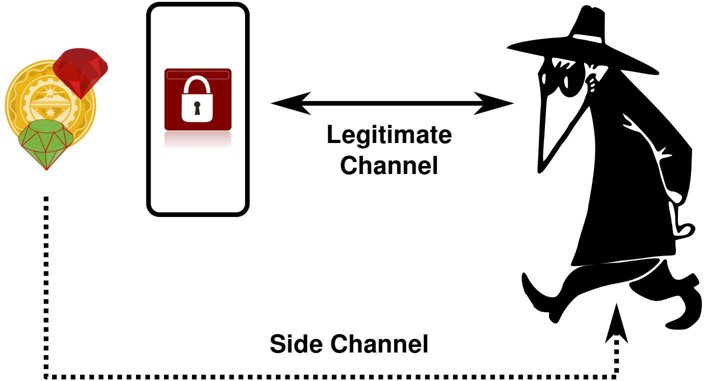
- Linking operational semantics and information theory.
- Testing the limits of pGCL in Isabelle.
- Hand calculation eliminated.
- Composes with L4.verified stack.

What's in the talk?

- What was the original result, and why formalise it?
- How was it formalised?
- What did we learn?

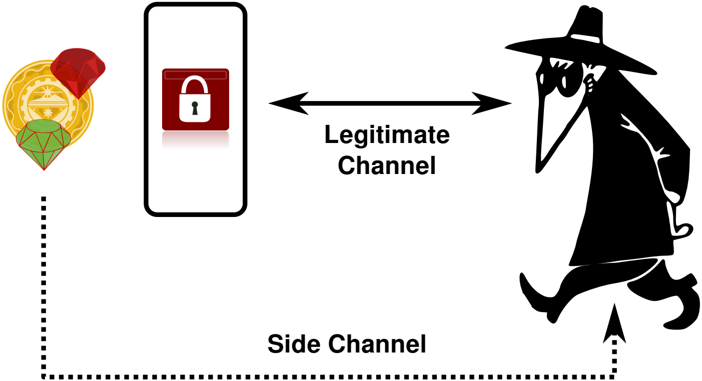
Outline

The Problem



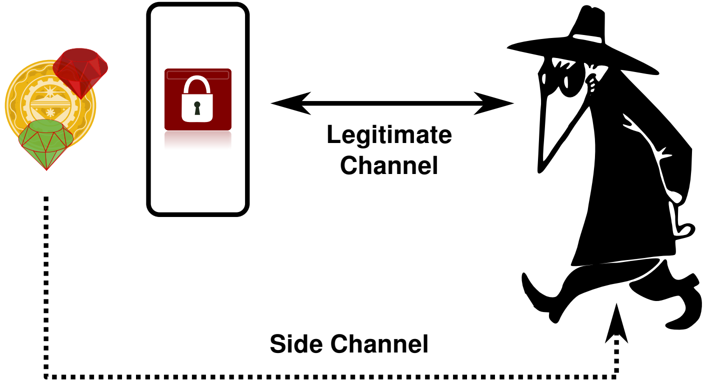
The attacker tries to guess the lock combination.

The Problem



After n tries he's locked out.

The Problem



Every guess leaks something about the combination.

Statistics

- The combination is **random**, and the attacker knows the distribution: $P(s)$.

Statistics

- The combination is **random**, and the attacker knows the distribution: $P(s)$.
- The side channel obeys $P(o|s)$.

Statistics

- The combination is **random**, and the attacker knows the distribution: $P(s)$.
- The side channel obeys $P(o|s)$.
- Given observations ol, the best guess is given by Bayes' rule:

$$P(s|ol) = \frac{P(ol|s)P(s)}{P(ol)}$$

Statistics

- The combination is **random**, and the attacker knows the distribution: $P(s)$.
- The side channel obeys $P(o|s)$.
- Given observations ol, the best guess is given by Bayes' rule:

$$P(s|ol) = \frac{P(ol, s)}{P(ol)}$$

Statistics

- The combination is **random**, and the attacker knows the distribution: $P(s)$.
- The side channel obeys $P(o|s)$.
- Given observations *ol*, the best guess is given by Bayes' rule:

$$P(s|ol) = \frac{P(ol, s)}{P(ol)}$$

- For the best guess, maximise $P(ol, s)$.

The Result

$$V \leq \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

The Result

$$\color{red}{V} \leq \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

Vulnerability is bounded above

The Result

$$V \leq \textcolor{red}{\sup_{\sigma}} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

Vulnerability is bounded above

…by the supremum over attack strategies

The Result

$$V \leq \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

Vulnerability is bounded above

...by the supremum over attack strategies

...of the sum over possible lists of observations

The Result

$$V \leq \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

Vulnerability is bounded above

...by the supremum over attack strategies

...of the sum over possible lists of observations

...and over the attacker's guesses

The Result

$$V \leq \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

Vulnerability is bounded above
…by the supremum over attack strategies
…of the sum over possible lists of observations
…and over the attacker's guesses
…of the joint probability of observations and guess.

Why Is This Important?

- An attacker that maximises $P(ol, s)$ is a worst-case scenario.

Why Is This Important?

- An attacker that maximises $P(ol, s)$ is a worst-case scenario.
- Such an attacker can be built (in theory) — That's the aim of machine learning. Therefore the bound is tight.

Why Is This Important?

- An attacker that maximises $P(ol, s)$ is a worst-case scenario.
- Such an attacker can be built (in theory) — That's the aim of machine learning. Therefore the bound is tight.
- We can safely assume that this is our adversary.

Why Is This Important?

- An attacker that maximises $P(ol, s)$ is a worst-case scenario.
- Such an attacker can be built (in theory) — That's the aim of machine learning. Therefore the bound is tight.
- We can safely assume that this is our adversary.

The Important Point

We didn't **assume** optimality, we **proved** it.

Why Formalise?

Why Formalise?

$$\sum_{s} P(s) * \left(\sum_{ol[..n]} \prod_{i=1}^{n} P(ol\,!(n-i)|s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail } i\ ol) \neq s) \right) =$$

$$\sum_{ol[..n]} \sum_{s} P(ol, s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail } i\ ol) \neq s)$$

Why Formalise?

$$\sum_{s} P(s) * \left(\sum_{ol[..n]} \prod_{i=1}^{n} P(ol\,!(n-i)|s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail } i\ ol) \neq s) \right) =$$

$$\sum_{ol[..n]} \sum_{s} P(ol, s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail } i\ ol) \neq s)$$

That's why.

Why Formalise?

$$\sum_{s} P(s) * \left(\sum_{ol[..n]} \prod_{i=1}^{n} P(ol\,!(n-i)|s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail}\ i\ ol) \neq s) \right) =$$

$$\sum_{ol[..n]} \sum_{s} P(ol, s) * \prod_{i=0}^{n} R(\sigma\ (\text{tail}\ i\ ol) \neq s)$$

That's why.

That's one of 40 or so delicate manipulations in the original proof — I ran out of whiteboard, and I don't trust my penmanship enough.

Why Formalise?

There are also a few more technically-justified reasons:

- Replace an ad-hoc operational model with a well-known formalism: pGCL.

Why Formalise?

There are also a few more technically-justified reasons:

- Replace an ad-hoc operational model with a well-known formalism: pGCL.
- It tested the limits of the pGCL formalisation.

Why Formalise?

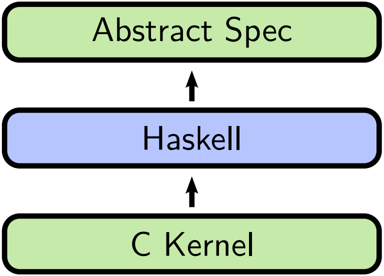
There are also a few more technically-justified reasons:

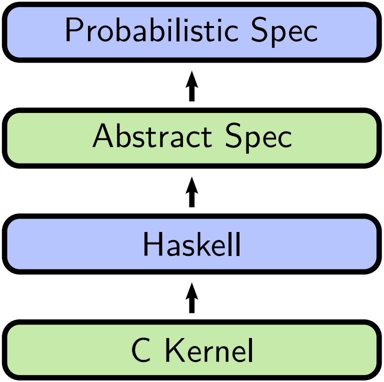
- Replace an ad-hoc operational model with a well-known formalism: pGCL.
- It tested the limits of the pGCL formalisation.
- We've shown before that the refinement order is compatible with L4.verified.

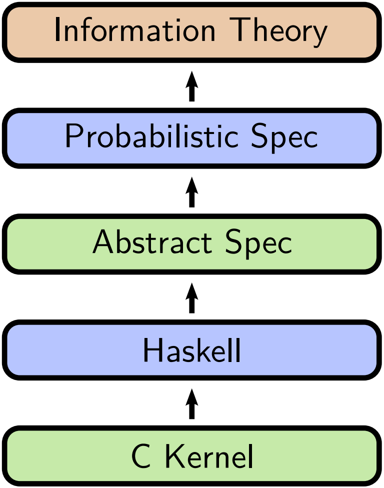
Why Formalise?

There are also a few more technically-justified reasons:

- Replace an ad-hoc operational model with a well-known formalism: pGCL.
- It tested the limits of the pGCL formalisation.
- We've shown before that the refinement order is compatible with L4.verified.
- This is a simple example. Scaling a paper proof is **hard**.


Abstract Spec
Haskell
C Kernel

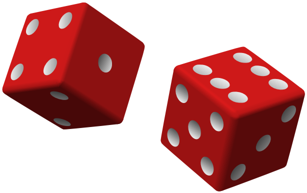

Probabilistic Spec
Abstract Spec
Haskell
C Kernel

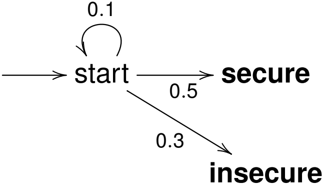

Information Theory
Probabilistic Spec
Abstract Spec
Haskell
C Kernel

Outline

- The Original Proof
- How It's Formalised and Why
- Outcomes

pGCL





- pGCL is a language of probabilistic automata.
- It models both demonic and probabilistic choice.
- We previously formalised it in Isabelle.

Why Guessing Attacks?

Security properties are often **hyperproperties**:

- Defined over sets of traces.
- Not preserved by refinement.

For a guessing attack, security is a property of the current state.

Security Predicate

Has the attacker guessed the secret yet?

Modelled as a loop:

$$\mathbf{do}\ g \neq s \longrightarrow \text{guess}$$

Termination

The Original Proof
How It's Formalised and Why
Outcomes

$$9, 9, 9, 9, 9, 9, 9, \ldots$$

$$\textbf{do } g \neq s \longrightarrow \text{guess}$$

Termination

$$9, 9, 9, 9, 9, 9, 9, \ldots$$

$$\textbf{do } g \neq s \longrightarrow \text{guess}$$

What happens if the loop doesn't terminate?

Termination

$$9, 9, 9, 9, 9, 9, 9, \ldots$$

$$\mathbf{do}\ g \neq s \longrightarrow \text{guess}$$

What happens if the loop doesn't terminate?

The probability of establishing the predicate (secure) is 0!

By default, nontermination acts the wrong way.

Partial Correctness

- The solution:

Partial Correctness

- The solution: the liberal (wlp) interpretation.

Partial Correctness

- The solution: the liberal (wlp) interpretation.
- “Correct if terminating”

Partial Correctness

- The solution: the liberal (wlp) interpretation.
- "Correct if terminating"
- A nonterminating program establishes any predicate with probability 1.

Partial Correctness

- The solution: the liberal (wlp) interpretation.
- "Correct if terminating"
- A nonterminating program establishes any predicate with probability 1.
- The probability of remaining secure is:

$$\text{wlp}\ \ (\textbf{do}\ g \neq s \longrightarrow \text{guess})\ \ \text{«secure»}$$

Refinement Soundness

- pGCL is a **probabilistic** logic.
- Refinement increases probabilities:

$$\frac{a \sqsubseteq b}{\text{wlp } a\ Q \vDash \text{wlp } b\ Q}$$

Refinement Soundness

NICTA

- pGCL is a **probabilistic** logic.
- Refinement increases probabilities:

$$\frac{a \sqsubseteq b}{\text{wlp } a\ Q \vDash \text{wlp } b\ Q}$$

Refinement preserves probabilistic security predicates.

Refinement Soundness

NICTA

- pGCL is a **probabilistic** logic.
- Refinement increases probabilities:

$$\frac{a \sqsubseteq b}{\text{wlp } a\ Q \vDash \text{wlp } b\ Q}$$

Refinement preserves probabilistic security predicates.

$$V_n = 1 - \text{wlp } \left(\textbf{do } g \neq s \longrightarrow \text{guess}\right) \text{ «secure»}$$

The Attack in pGCL

$$\textbf{do } g \neq s \longrightarrow \text{guess}$$

Guess until we get the secret, ...

The Attack in pGCL

$\textbf{choose } s \textbf{ at } P(s)$

$\textbf{do } g \neq s \longrightarrow \text{guess}$

...which is chosen randomly.

The Attack in pGCL

$\textbf{choose } s \textbf{ at } P(s)$

$\textbf{do } g \neq s \longrightarrow \text{guess}$

Every guess leaks an observation.

The Attack in pGCL

NICTA

The Original Proof

How It's Formalised and Why

Outcomes

$$\textbf{choose } s \textbf{ at } P(s)$$
$$\textbf{do } g \neq s \longrightarrow$$
$$\quad \textbf{bind } o \textbf{ at } P(o|s) \text{ in}$$
$$\quad ol := o{:}ol$$

Every guess leaks an observation.

The Attack in pGCL

$ol := [\,]$

choose s **at** $P(s)$

do $g \neq s \longrightarrow$

  bind o **at** $P(o|s)$ in

  $ol := o{:}ol$

Initially, there are none.

The Attack in pGCL

$$
\begin{array}{l}
ol := [\,] \\
\textbf{choose } s \textbf{ at } P(s) \\
\textbf{do } {\color{red} g} \neq s \longrightarrow \\
\qquad \textbf{bind } o \textbf{ at } P(o|s) \text{ in} \\
\qquad ol := o{:}ol
\end{array}
$$

The attacker uses some strategy, σ . . .

The Attack in pGCL

$$
\begin{array}{l}
ol := [\,] \\
\textbf{choose } s \textbf{ at } P(s) \\
\textbf{do } \sigma\ ol \neq s \longrightarrow \\
\qquad \textbf{bind } o \textbf{ at } P(o|s) \text{ in} \\
\qquad ol := o{:}ol
\end{array}
$$

The attacker uses some strategy, σ ...

The Attack in pGCL

any σ

$ol := [\,]$

choose s **at** $P(s)$

do σ $ol \neq s \longrightarrow$

  bind o **at** $P(o|s)$ in

  $ol := o{:}ol$

…which is freely chosen, but may **not** depend on s.

The Attack in pGCL

$$
\begin{array}{l}
\textbf{any } \sigma \\
ol := [\,] \\
\textbf{choose } s \textbf{ at } P(s) \\
\textbf{do } \sigma\ ol \neq s \longrightarrow \\
\qquad \textbf{bind } o \textbf{ at } P(o|s) \text{ in} \\
\qquad ol := o{:}ol
\end{array}
$$

. . . which is freely chosen, but may **not** depend on *s*.

The Security Predicate

NICTA

What does it mean to terminate in a secure state?
The attacker has used all n guesses, without guessing correctly:

$$n < |ol| \wedge \forall i \leq n.\ \sigma\ (\text{tail } i\ ol) \neq s$$

The Security Predicate

What does it mean to terminate in a secure state?
The attacker has used all n guesses, without guessing correctly:

$$n < |ol| \wedge \forall i \leq n.\ \sigma\ (\text{tail}\ i\ ol) \neq s$$

The Security Predicate

What does it mean to terminate in a secure state?
The attacker has used all n guesses, without guessing correctly:

$$n < |ol| \wedge \forall i \leq n.\ \sigma\ (\text{tail}\ i\ ol) \neq s$$

The Security Predicate

What does it mean to terminate in a secure state?
The attacker has used all n guesses, without guessing correctly:

$$n < |ol| \wedge \forall i \leq n.\ \sigma\ (\mathrm{tail}\ i\ ol) \neq s$$

Probabilistic Security Predicate

What is the **probability** that we end in a secure state?

Classical Invariants

In a classical logic, we annotate loops using **invariants**:

$$\frac{\{I \wedge G\}\ \text{body}\ \{I\}}{\{I\}\ \textbf{do}\ G \rightarrow \text{body}\ \{I \wedge \neg G\}}$$

Classical Invariants

In a classical logic, we annotate loops using **invariants**:

$$\frac{\{I \wedge G\}\ \text{body}\ \{I\}}{\{I\}\ \textbf{do}\ G \rightarrow \text{body}\ \{I \wedge \neg G\}}$$

A classical invariant becomes 'more true':

$$\frac{G\ s}{I\ s \longrightarrow \text{wlp body}\ I\ s}$$

Probabilistic Invariants

Probabilistic loops are almost exactly equivalent:

$$\frac{\{I \,\&\&\, «G»\}\ \text{body}\ \{I\}}{\{I\}\ \textbf{do}\ G \rightarrow \text{body}\ \{I \,\&\&\, «\neg G»\}}$$

Probabilistic Invariants

Probabilistic loops are almost exactly equivalent:

$$\frac{\{I \mathbin{\&\&} «G»\}\ \text{body}\ \{I\}}{\{I\}\ \textbf{do}\ G \rightarrow \text{body}\ \{I \mathbin{\&\&} «\neg G»\}}$$

A **probabilistic** invariant gets 'bigger':

$$\frac{G\ s}{I\ s \leq \text{wlp body}\ I\ s}$$

The Loop Invariant

$$I = \prod_{i=0}^{n \sqcap |ol|} «\sigma\ (\text{tail}\ i\ ol) \neq s»$$
$$* \sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' \mathbin{@} ol)\,!(n-i)|s)\ * \\ \quad «\sigma\ (\text{tail}\ i\ (ol' \mathbin{@} ol)) \neq s» \end{array} \right)$$

This consists of two parts:

The Loop Invariant

$$I = \prod_{i=0}^{n \sqcap |ol|} «\sigma\ (\text{tail}\ i\ ol) \neq s»$$

$$* \sum_{ol' [..n-|ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' @ ol)\,!(n-i)|s) \; * \\ \quad «\sigma\ (\text{tail}\ i\ (ol' @ ol)) \neq s» \end{array} \right)$$

This consists of two parts:

- Whether the predicate holds **in the past**.

The Loop Invariant

$$I = \prod_{i=0}^{n \sqcap |ol|} «\sigma\ (\text{tail}\ i\ ol) \neq s»$$

$$* \sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' \mathbin{@} ol)\,!(n-i)|s) * \\ \quad «\sigma\ (\text{tail}\ i\ (ol' \mathbin{@} ol)) \neq s» \end{array} \right)$$

This consists of two parts:

- Whether the predicate holds **in the past**.
- The **probability** that it will continue to hold.

The Postcondition

NICTA

The Original Proof

How It's Formalised and Why

Outcomes

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

The Postcondition

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } \left(\textbf{do } g \neq s \longrightarrow \text{guess}\right) \; I \mathbin{\&\&} «g = s»$$

The Postcondition

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \ I \mathbin{\&\&} «g = s»$$

The Original Proof
How It's Formalised and Why
Outcomes

The Postcondition

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \; I \mathbin{\&\&} «g = s»$$

Also by evaluation:

$$I \mathbin{\&\&} «g = s» \vDash «\text{secure}»$$

The Postcondition

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \;\; I \mathbin{\&\&} «g = s»$$

Also by evaluation:

$$I \mathbin{\&\&} «g = s» \vDash «\text{secure}»$$

The Original Proof

How It's Formalised and Why

Outcomes

The Postcondition

I is an invariant:

$$I \,\&\&\, «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \; I \,\&\&\, «g = s»$$

Also by evaluation:

$$I \,\&\&\, «g = s» \vDash «\text{secure}»$$

Thus finally:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \; «\text{secure}»$$

The Postcondition

I is an invariant:

$$I \mathbin{\&\&} «g \neq s» \vDash \text{wlp guess } I$$

Hence by the loop rule:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \;\; I \mathbin{\&\&} «g = s»$$

Also by evaluation:

$$I \mathbin{\&\&} «g = s» \vDash «\text{secure}»$$

Thus finally:

$$I \vDash \text{wlp } (\textbf{do } g \neq s \longrightarrow \text{guess}) \;\; «\text{secure}»$$

The Precondition

$$\prod_{i=0}^{n \sqcap |ol|} \text{«}\sigma\ (\text{tail}\ i\ ol) \neq s\text{»} *$$

$$\sum_{ol' [..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' @ ol)\ !(n-i)|s) * \\ \quad \text{«}\sigma\ (\text{tail}\ i\ (ol' @ ol)) \neq s\text{»} \end{pmatrix}$$

---

do $g \neq s \rightarrow$ guess

The Precondition

$$\prod_{i=0}^{n \sqcap |ol|} «\sigma\ (\text{tail}\ i\ ol) \neq s» *$$

$$\sum_{ol' [..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' @ ol)\,!(n-i)|s) * \\ \quad «\sigma\ (\text{tail}\ i\ (ol' @ ol)) \neq s» \end{pmatrix}$$

$ol := [\,]\ ;;$

do $g \neq s \rightarrow$ guess

The Precondition

$$\prod_{i=0}^{0} «\sigma\ (\text{tail}\ i\ ol) \neq s» *$$

$$\sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' \mathbin{@} ol)\,!(n-i)|s) * \\ \quad «\sigma\ (\text{tail}\ i\ (ol' \mathbin{@} ol)) \neq s» \end{array} \right)$$

$ol := [\,]\ ;;$

$\textbf{do}\ g \neq s \rightarrow \text{guess}$

The Precondition

$$\sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' \mathbin{@} ol)\,!(n-i)|s) \; * \\ \quad «\sigma \; (\text{tail } i \; (ol' \mathbin{@} ol)) \neq s» \end{pmatrix}$$

$ol := [\,] \,;;$

$\textbf{do } g \neq s \rightarrow \text{guess}$

The Precondition

$$\sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' \mathbin{@} ol)\,!(n-i)|s) * \\ \quad «\sigma\ (\mathrm{tail}\ i\ (ol' \mathbin{@} ol)) \neq s» \end{pmatrix}$$

---

choose s **at** $P(s)$; ;

$ol := [\,]$; ;

do $g \neq s \rightarrow$ guess

The Precondition

$$\sum_{s} P(s) * \sum_{ol' [..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' @ ol) \,!(n-i)|s) * \\ «\sigma \; (\text{tail } i \; (ol' @ ol)) \neq s» \end{pmatrix}$$

---

choose s **at** $P(s)$; ;

$ol := [\,]$; ;

do $g \neq s \rightarrow$ guess

The Precondition

$$\sum_{s} P(s) * \sum_{ol' [..n - |ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' \mathbin{@} ol) \mathbin{!} (n-i) | s) * \\ \quad «\sigma \text{ (tail } i \text{ } (ol' \mathbin{@} ol)) \neq s» \end{array} \right)$$

---

any σ ; ;
choose s **at** $P(s)$; ;
$ol := [\,]$; ;
do $g \neq s \rightarrow$ guess

The Precondition

$$\inf_{\sigma} \sum_{s} P(s) * \sum_{ol'[..n-|ol|]} \prod_{i=|ol|+1}^{n} \begin{pmatrix} P((ol' \mathbin{@} ol)\,!(n-i)|s) * \\ «\sigma\ (\text{tail } i\ (ol' \mathbin{@} ol)) \neq s» \end{pmatrix}$$

---

any σ ; ;

choose s **at** $P(s)$; ;

$ol := [\,]$; ;

do $g \neq s \rightarrow$ guess

The Precondition

$$\inf_{\sigma} \sum_{s} P(s) * \sum_{ol' [..n-|ol|]} \prod_{i=|ol|+1}^{n} \left(\begin{array}{l} P((ol' \mathbin{@} ol) \,!(n-i) | s) \; * \\ \quad «\sigma \; (\mathrm{tail}\; i \; (ol' \mathbin{@} ol)) \neq s» \end{array} \right)$$

---

any σ ; ;
choose s **at** $P(s)$; ;
$ol := [\,]$; ;
do $g \neq s \rightarrow$ guess

The Precondition

$$\inf_{\sigma} \left(1 - \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s) \right)$$

$$\begin{aligned}
&\textbf{any } \sigma \,;; \\
&\textbf{choose } s \textbf{ at } P(s) \,;; \\
&ol := [\,] \,;; \\
&\textbf{do } g \neq s \to \text{guess}
\end{aligned}$$

The Precondition

$$V_n = 1 - \inf_{\sigma} \left(1 - \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s) \right)$$

$$\begin{array}{l}
\textbf{any } \sigma\ ;; \\
\textbf{choose } s \textbf{ at } P(s)\ ;; \\
ol := [\,]\ ;; \\
\textbf{do } g \neq s \rightarrow \text{guess}
\end{array}$$

The Precondition

$$V_n = 1 - \inf_{\sigma} \left(1 - \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s) \right)$$

$$\begin{array}{l} \textbf{any}\ \sigma\ ;; \\ \textbf{choose}\ s\ \textbf{at}\ P(s)\ ;; \\ ol := [\,]\ ;; \\ \textbf{do}\ g \neq s \rightarrow \text{guess} \end{array}$$

The Original Proof

How It's Formalised and Why

Outcomes

The Precondition

$$V_n = \color{red}{\sup_\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

---

any σ ; ;
choose s **at** $P(s)$; ;
$ol := [\,]$; ;
do $g \neq s \rightarrow$ guess

The Precondition

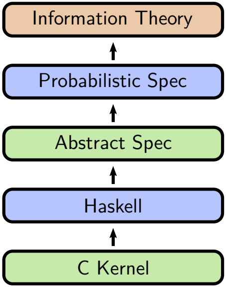
$$V_n = \sup_{\sigma} \sum_{ol[..n]} \sum_{s \in \Gamma\ \sigma\ ol} P(ol, s)$$

$$\begin{array}{l} \textbf{any}\ \sigma\ ;; \\ \textbf{choose}\ s\ \textbf{at}\ P(s)\ ;; \\ ol := [\,]\ ;; \\ \textbf{do}\ g \neq s \rightarrow \text{guess} \end{array}$$

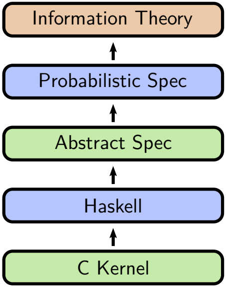
Outline

- The Original Proof
- How It's Formalised and Why
- Outcomes

A Verified Chain From C to Info Theory

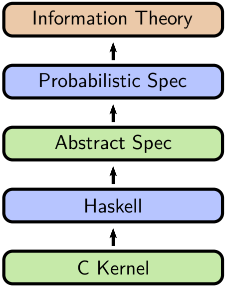


A Verified Chain From C to Info Theory



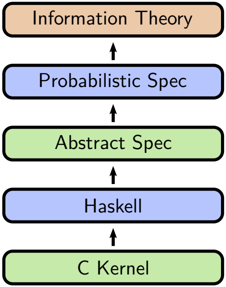
- We've shown that we can embed seL4 into a probabilistic logic.

A Verified Chain From C to Info Theory



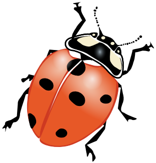
- We've shown that we can embed seL4 into a probabilistic logic.
- Now there's another step: quantitative information flow.

A Verified Chain From C to Info Theory



- We've shown that we can embed seL4 into a probabilistic logic.
- Now there's another step: quantitative information flow.
- Vulnerability is preserved by refinement all the way down.

There Were No Proof Bugs



The pen-and-paper proof was correct.
That's great for my self-confidence, but makes this slide rather dull. ;)

Improving the Formalisation

We did have to make some changes to the existing formalisation:

Improving the Formalisation

We did have to make some changes to the existing formalisation:

- The existing VCG wasn't powerful enough.

Improving the Formalisation

We did have to make some changes to the existing formalisation:

- The existing VCG wasn't powerful enough.
- We had to fully treat recursion — It is now as powerful as the published results.

Improving the Formalisation

We did have to make some changes to the existing formalisation:

- The existing VCG wasn't powerful enough.
- We had to fully treat recursion — It is now as powerful as the published results.
- The theory is now in a usable state — Under submission to AFP.

Summary

Summary

- We've shown how to formally verify a probabilistic property,

The Original Proof
How It's Formalised and Why
Outcomes

Summary

- We've shown how to formally verify a probabilistic property,

  ...that is preserved by refinement,

Summary

- We've shown how to formally verify a probabilistic property,

  . . . that is preserved by refinement,

  . . . reusing a real, large-scale proof.

Summary

- We've shown how to formally verify a probabilistic property,

  ...that is preserved by refinement,

  ...reusing a real, large-scale proof.
- This particular result can be instantiated with a model for $P(o|s)$ (c.f. my thesis).

Summary

- We've shown how to formally verify a probabilistic property,

  . . . that is preserved by refinement,

  . . . reusing a real, large-scale proof.
- This particular result can be instantiated with a model for $P(o|s)$ (c.f. my thesis).
- The approach can also be used for any state-based probabilistic property of the correct form.

Questions?