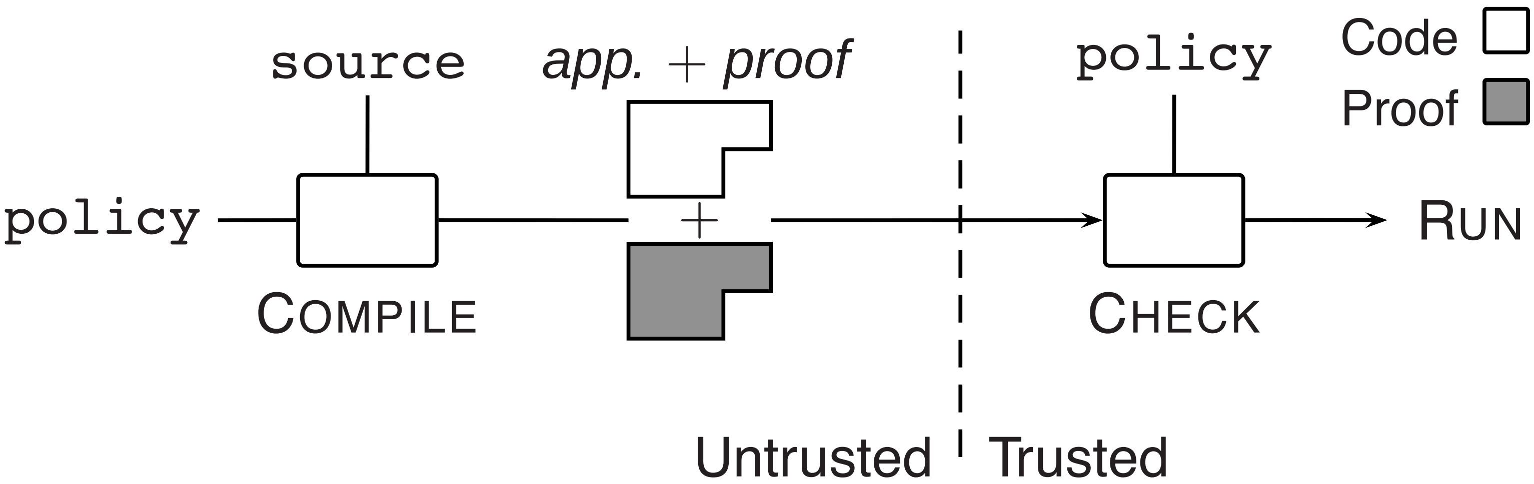
Reference monitors for proof-carrying code

Simon Winwood
University of New South Wales, National ICT Australia
sjw@cse.unsw.edu.au

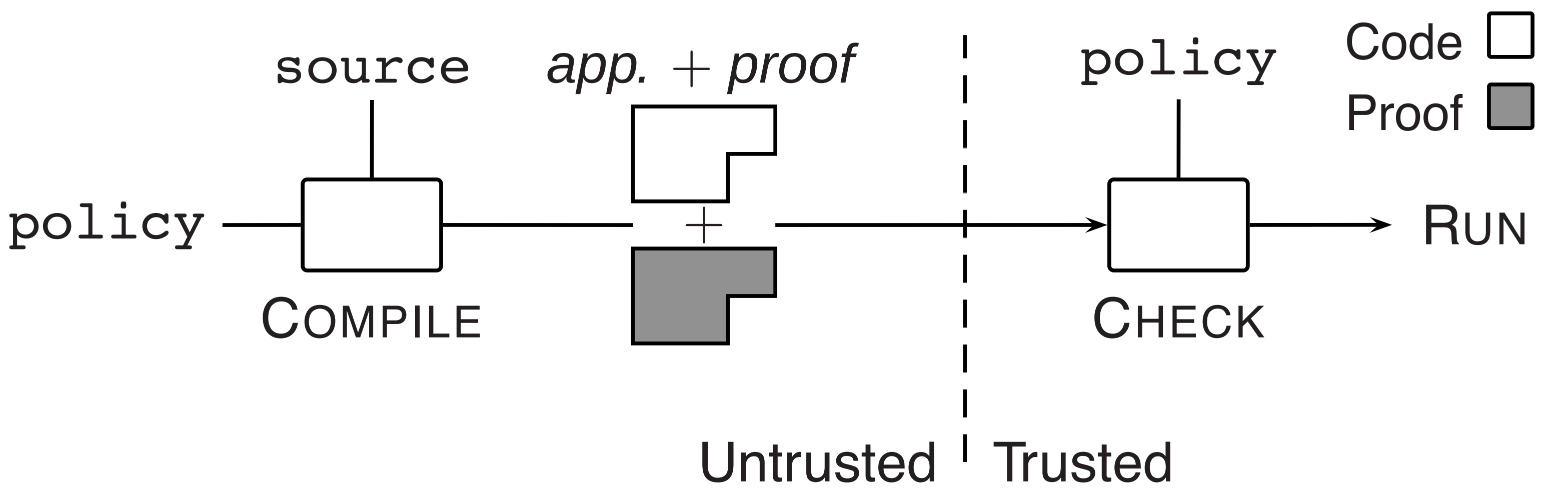
Proof-carrying code



Proof-carrying code:

- ➜ Application carries a machine-checkable safety proof
  $\longrightarrow$ Need only trust checker (+ semantics, policy, logic, ...)
- ➜ Proofs generated automatically
- ➜ Compiler uses language properties to show memory + control safety
  $\longrightarrow$ OK for simple code, e.g. filters

PROOF-CARRYING CODE



Proof-carrying code:

- ➜ Application carries a machine-checkable safety proof
  - → Need only trust checker (+ semantics, policy, logic, ...)
- ➜ Proofs generated automatically
- ➜ Compiler uses language properties to show memory + control safety
  - → OK for simple code, e.g. filters

Example apps — extensible web browsers, kernels, ...

What about high-level policies?:

- ➜ Can reason about system calls, library invocations, etc.
- ➜ Can reason about history of invocations

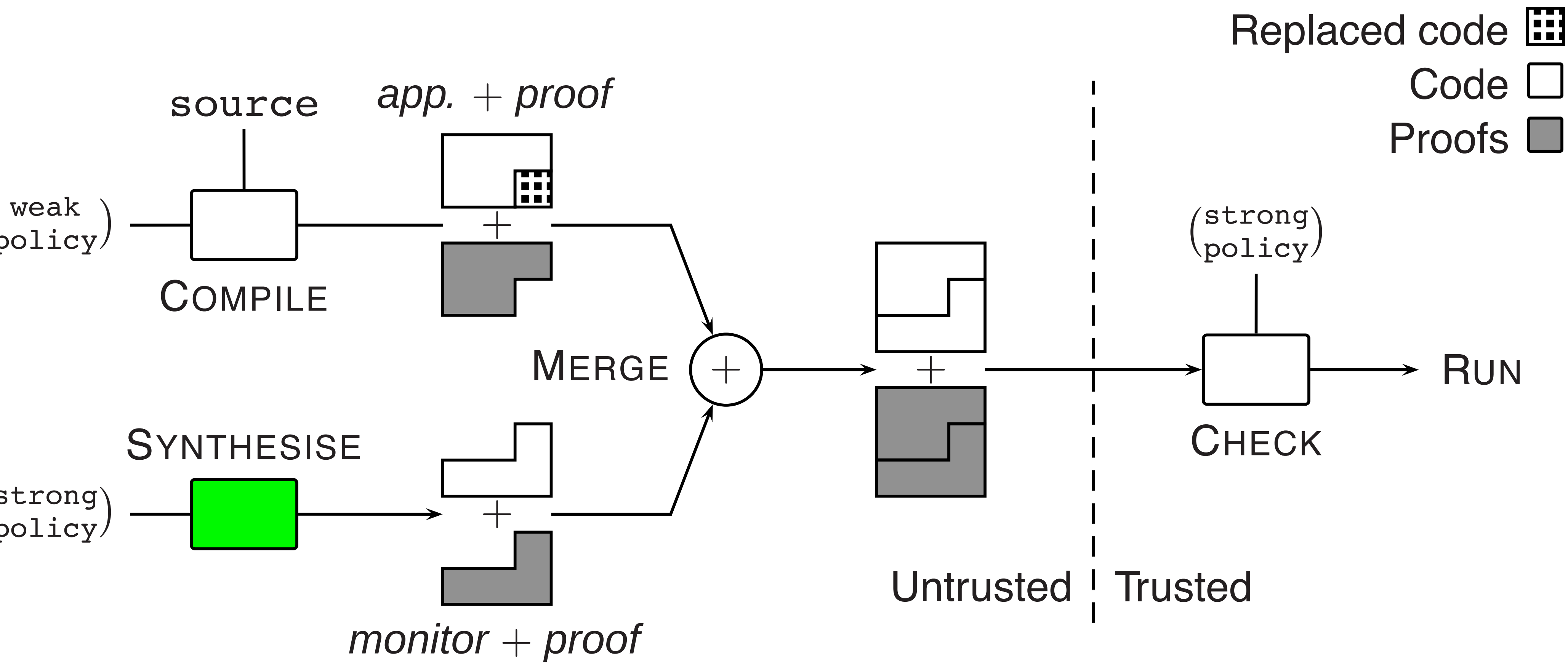
What about high-level policies?:

- ➜ Can reason about system calls, library invocations, etc.
- ➜ Can reason about history of invocations

Example: Chinese Wall Policy

> *A user may read/write files for any client, but once they have done so, they must not access files belonging to other clients.*

My work



Synthesise a reference monitor + proofs from security policy ψ

$\rightarrow$ monitor performs sensitive operations on behalf of application.

Past-time propositional temporal logic (P3TL)

P3TL is the safety fragment of propositional linear temporal logic

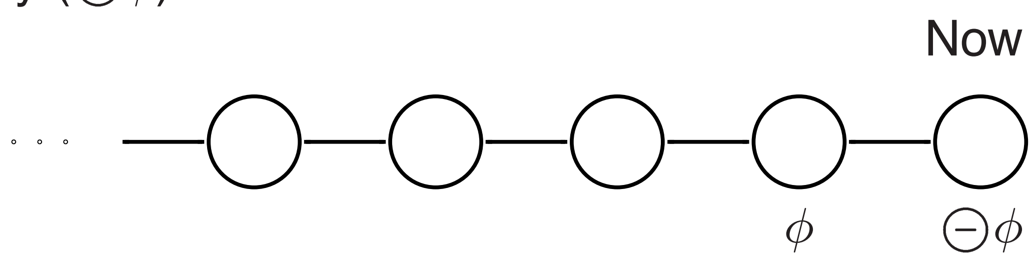
- propositional variables: $p, q, \ldots$
- usual propositional connective: $\rightarrow, \neg, \wedge, \vee, \ldots$
- previously ($\ominus\phi$)

- since ($\phi S \psi$)

Past-time propositional temporal logic (P3TL)

P3TL is the safety fragment of propositional linear temporal logic

- propositional variables: $p, q, \ldots$
- usual propositional connective: $\rightarrow, \neg, \wedge, \vee, \ldots$
- previously ($\ominus\phi$)

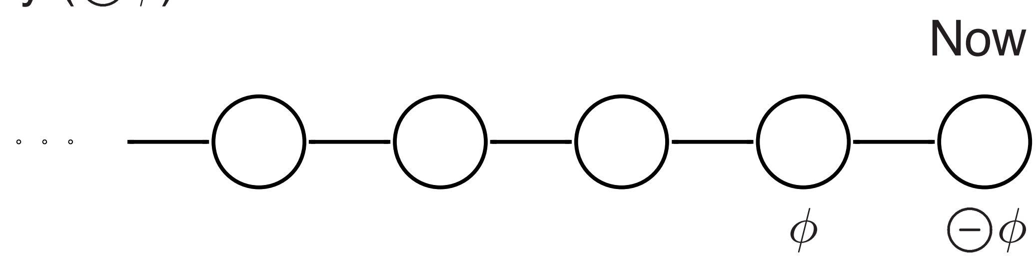


- since ($\phi S \psi$)

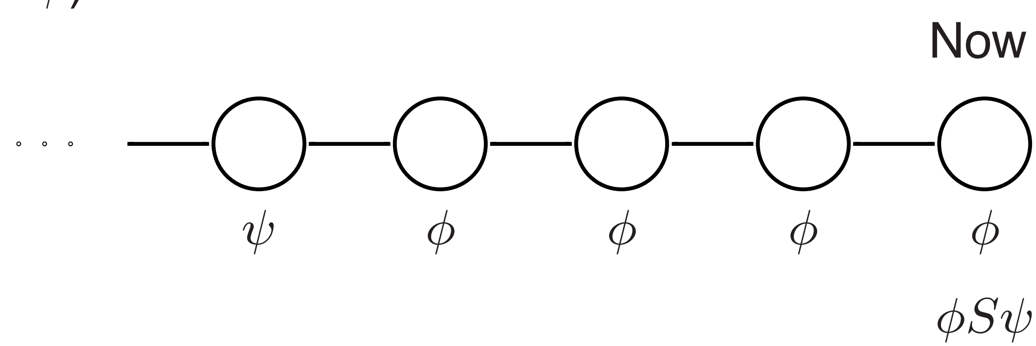
Past-time propositional temporal logic (P3TL)

P3TL is the safety fragment of propositional linear temporal logic

- propositional variables: $p, q, \ldots$
- usual propositional connective: $\rightarrow, \neg, \wedge, \vee, \ldots$
- previously ($\ominus\phi$)



- since ($\phi S \psi$)



P3TL (cont.):

Chinese Wall in P3TL:

$$(\mathtt{access}(f) \ \wedge \ f \in \textbf{Coke}) \longrightarrow (\neg \diamondminus(\mathtt{access}(g) \ \wedge \ g \in \textbf{Pepsi}))$$

$$\wedge$$

$$(\mathtt{access}(h) \ \wedge \ h \in \textbf{Pepsi}) \longrightarrow (\neg \diamondminus(\mathtt{access}(k) \ \wedge \ k \in \textbf{Coke}))$$

where

$$\begin{array}{rcll} \mathtt{init} & \equiv & \neg \ominus \top & \text{(initial state)} \\ \diamondminus \psi & \equiv & \top S \psi & \text{(at some point previously)} \end{array}$$

A LOGIC FOR MONITORS

P3TL gives us a logic for policies (sequences of worlds)

Use a Hoare-like logic (in Isabelle/HOL) for programs

$$
\begin{aligned}
s \;::=\; & x \;:=\; e \mid \texttt{IF}\ e\ \texttt{THEN}\ s\ \texttt{ELSE}\ s \\
& \mid \texttt{WHILE}\ e\ \texttt{DO}\ s \mid s;\ s \mid \texttt{Secure}\ \phi
\end{aligned}
$$

➜ Models are a tuple of (program state, state trace)

$\longrightarrow$ trace does not appear at runtime

➜ Use a shallow embedding for the assertion logic

$\longrightarrow$ assertions are predicates on models

➜ Abstract over nature of secure events with `Secure` statement

$\longrightarrow$ update world sequence after this statement

A Hoare Logic: Rules:

$$\vdots$$

$$\frac{\vdash \{\lambda(s,\sigma).P(s,\sigma) \wedge b\ s\}e\{Q\} \quad \vdash \{\lambda(s,\sigma).P(s,\sigma) \wedge \neg b\ s\}e'\{Q\}}{\vdash \{P\}\mathtt{IF}\ b\ \mathtt{THEN}\ e\ \mathtt{ELSE}\ e'\{Q\}}\ \text{IF}$$

$$\vdots$$

$$\frac{\forall(s,\sigma).(P(s,\sigma) \longrightarrow (\sigma \cdot s \models \phi)) \quad \forall(s,\sigma).P(s,\sigma) \longrightarrow Q(s,\sigma \cdot s)}{\vdash \{P\}\mathtt{Secure}\ \phi\{Q\}}\ \text{SECURE}$$

A Hoare Logic: Rules:

$$\dfrac{\begin{array}{c}\vdots \\ \vdash \{\lambda(s,\sigma).P(s,\sigma) \wedge b\ s\}e\{Q\} \quad \vdash \{\lambda(s,\sigma).P(s,\sigma) \wedge \neg b\ s\}e'\{Q\}\end{array}}{\vdash \{P\}\texttt{IF}\ b\ \texttt{THEN}\ e\ \texttt{ELSE}\ e'\{Q\}}\ \textsf{IF}$$

$$\dfrac{\begin{array}{c}\vdots \\ \forall(s,\sigma).(P(s,\sigma) \longrightarrow (\sigma \cdot s \vDash \phi)) \quad \forall(s,\sigma).P(s,\sigma) \longrightarrow Q(s,\sigma \cdot s)\end{array}}{\vdash \{P\}\texttt{Secure}\ \phi\{Q\}}\ \textsf{SECURE}$$

The premises to SECURE are our synthesis proof obligations

SYNTHESIS

Basic idea

- ➜ Keep track of values of temporal sub-formula in invariant
  $\longrightarrow$ only need for the previous world
- ➜ When we see a $\ominus\phi$ check sub-formula state variable for ϕ
- ➜ Unfold $\phi S \psi$ to $\psi \vee (\ominus(\phi S \psi) \wedge \phi)$

SYNTHESIS

Basic idea

- ➜ Keep track of values of temporal sub-formula in invariant
  $\longrightarrow$ only need for the previous world
- ➜ When we see a $\ominus\phi$ check sub-formula state variable for ϕ
- ➜ Unfold $\phi S \psi$ to $\psi \vee (\ominus(\phi S \psi) \wedge \phi)$

PATHEXPLORER (Havelund and Rosu, TACAS'02) uses a similar algorithm (sans proofs)

SYNTHESIS

Basic idea

- ➜ Keep track of values of temporal sub-formula in invariant
  $\longrightarrow$ only need for the previous world
- ➜ When we see a $\ominus\phi$ check sub-formula state variable for ϕ
- ➜ Unfold $\phi S \psi$ to $\psi \vee (\ominus(\phi S \psi) \wedge \phi)$

PATHEXPLORER (Havelund and Rosu, TACAS'02) uses a similar algorithm (sans proofs)

Building the proof

- ➜ Monitor invariant: $b_1 \equiv \ominus\psi_1 \wedge \ldots b_n \equiv \ominus\psi_n$
  $\longrightarrow$ ψ_i are sub-formulae
- ➜ Proof obligations combine checks and invariant.
- ➜ Each step in synthesis adds proof rule to obligations..

A monitor for $\ominus x = 1 \rightarrow (x < 5)\,\mathcal{S}\,(y = 1)$:

$$\texttt{inv} \equiv \texttt{state}_0 \leftrightarrow \ominus x = 1 \wedge \texttt{state}_1 \leftrightarrow \ominus((x < 5)\,\mathcal{S}\,(y = 1))$$

```
IF (x = 1) THEN
  tmp0 := true
ELSE
  tmp0 := false
IF (y = 1) THEN
  tmp1 := true
ELSE
  IF (state1 = 1) THEN
    IF (x < 5) THEN
      tmp1 := true        (*)
    ELSE
      tmp1 := false
  ELSE
    tmp1 := false
IF (state0 = true) THEN
  IF (tmp1 = true) THEN
    Secure (policy);
    state0 := tmp0;
    state1 := tmp1
  ELSE
    Error
ELSE
  Secure (policy);
  state0 := tmp0;
  state1 := tmp1
```

Proof of

$$\texttt{tmp}_1 \leftrightarrow s \models (x < 5)\,\mathcal{S}\,(y = 1)$$

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{inv \qquad state_1}{s \models \ominus(p\,\mathcal{S}\,q)}\,\textsc{Inv-}\pi \qquad \dfrac{p}{s \not\models \neg p}\,\textsc{NNegI}}{s \not\models \ominus(p\,\mathcal{S}\,q) \rightarrow \neg p}\,\textsc{NImplI}_2}{s \models \neg(\ominus(p\,\mathcal{S}\,q) \rightarrow \neg p)}\,\textsc{NegI}}{s \models \neg q \rightarrow (\ominus(p\,\mathcal{S}\,q) \rightarrow \neg p)}\,\textsc{ImplI}_1}{s \models p\,\mathcal{S}\,q}\,\textsc{SinceI}$$

FIRST-ORDER POLICY LOGICS

What about:

$$\mathtt{read}(X) \longrightarrow (\neg\mathtt{close}(X))\,\mathcal{S}\,\mathtt{open}(X)$$

Problem: X spans the temporal connective
$\rightarrow$ not a valid P3TL formula

First-order policy logics

What about:

$$\mathtt{read}(X) \longrightarrow (\neg\mathtt{close}(X))\,\mathcal{S}\,\mathtt{open}(X)$$

Problem: X spans the temporal connective
$\rightarrow$ not a valid P3TL formula

Solution: Move to a first-order logic:

$$\forall X.(\mathtt{read}(X) \longrightarrow (\neg\mathtt{close}(X))\,\mathcal{S}\,\mathtt{open}(X))$$

First-order policy logics

What about:

$$\mathtt{read}(X) \longrightarrow (\neg\mathtt{close}(X))\,\mathcal{S}\,\mathtt{open}(X)$$

Problem: X spans the temporal connective
$\rightarrow$ not a valid P3TL formula

Solution: Move to a first-order logic:

$$\forall X.(\mathtt{read}(X) \longrightarrow (\neg\mathtt{close}(X))\,\mathcal{S}\,\mathtt{open}(X))$$

Non-trivial — FOLTL is much less well-behaved than PLTL

Current work:

- ➜ Past-time only is decidable
- ➜ Naive monitors can be very expensive
- ➜ How to do this efficiently?

Current work:

- ➜ Past-time only is decidable
- ➜ Naive monitors can be very expensive
- ➜ How to do this efficiently?

Synthesis is similar to propositional case

- ➜ Require a set of satisfying values for each formula (not 1 bit)
  $\longrightarrow$ size can be linear in length of trace!
- ➜ Each step refines possible values
- ➜ Existentials check non-emptiness of sets

QUESTIONS?