(Hyper)Property-Preserving Compilers

summer semester 18-19, block

Marco Patrignani[1,2]

Stanford University

CISPA HELMHOLTZ CENTER FOR INFORMATION SECURITY

Properties and Hyperproperties

- Formalise any security property
- Established theory with practical applications

Recommended reading:

- Schneider. 2000. Enforceable security policies.
- Alpern and Schneider. 1985. Defining liveness.
- Clarkson and Schneider. 2010. Hyperproperties.

Security Properties

- Property = set of traces that respect a certain condition (or predicate)

Security Properties

- Property = set of traces that respect a certain condition (or predicate)
- trace t = sequence of (formalised as $\overline{sth}$)

Security Properties

- Property = set of traces that respect a certain condition (or predicate)
- trace t = sequence of (formalised as $\overline{sth}$)
    - program states Θ

Security Properties

- Property = set of traces that respect a certain condition (or predicate)
- trace t = sequence of (formalised as $\overline{sth}$)
    - program states Θ
    - component-context interactions $\alpha?\alpha!\cdots$

Security Properties

- Property = set of traces that respect a certain condition (or predicate)
- trace t = sequence of (formalised as $\overline{sth}$)
    - program states Θ
    - component-context interactions $\alpha?\alpha!\cdots$
    - code-environment interaction $read\ v; write\ v$

We use t abstractly now, though mostly:
$t = \overline{\Theta}$

Security Properties

- A Trace captures a single run of a program

Security Properties

- A Trace captures a single run of a program
- A Set of traces captures all individual runs of any program

Security Properties

- A Trace captures a single run of a program
- A Set of traces captures all individual runs of any program

This is unlike program equivalence:

- properties talk a single program

Examples

- NRW: $\{t \mid \nexists \Theta < \Theta'.\ \vdash read\Theta \wedge \vdash send\Theta'\}$
  NRW: *the program does not send on the network after reading a file*

Examples

- NRW: $\{t \mid \nexists \Theta < \Theta'.\ \vdash read\Theta \wedge \vdash send\Theta'\}$
  NRW: *the program does not send on the network after reading a file*
  $\vdash read\Theta$ and $\vdash send\Theta'$ are abstract predicates

Examples

- NRW: $\{t \mid \nexists \Theta < \Theta'. \vdash read\Theta \land \vdash send\Theta'\}$
  NRW: *the program does not send on the network after reading a file*
  $\vdash read\Theta$ and $\vdash send\Theta'$ are abstract predicates
- GS: $\{t \mid \vdash req\Theta_i \Rightarrow \vdash resp\Theta_j$ where $j > i\}$
  GS: *the program eventually responds to the requests*

Safety and Liveness

Properties are partitioned in

- Safety: something bad does not happen (NRW)

Safety and Liveness

Properties are partitioned in

- Safety: something bad does not happen (NRW)
- Liveness: something good eventually happens (GS)

Safety

- Safety = integrity

Safety

- Safety = integrity
- Safety ≠ confidentiality

Safety

- Safety = integrity
- Safety $\neq$ confidentiality
- but, Safety = weak secrecy: we don't leak a fresh k to $\mathfrak{C}$

Safety as a dual

- Take the traces that define a safety property

Safety as a dual

- Take the traces that define a safety property
- Describe safety by the so-called set of bad prefixes

Safety as a dual

- Take the traces that define a safety property
- Describe safety by the so-called set of bad prefixes
- In the following: m is a finite trace t (a finite $\overline{\Theta}$) aka a prefix

Safety as a dual

- Take the traces that define a safety property
- Describe safety by the so-called set of bad prefixes
- In the following: m is a finite trace t (a finite $\overline{\Theta}$) aka a prefix
- NRW-dual:
  $\{m \mid \Theta < \Theta'.\ \vdash read\Theta \wedge \vdash send\Theta'\}$

Beyond Properties: Hyperproperties

- Properties = sets of traces

Beyond Properties: Hyperproperties

- Properties = sets of traces
- capture a single run (the trace) of any program (the set)

Beyond Properties: Hyperproperties

- Properties = sets of traces
- capture a single run (the trace) of any program (the set)

  Hyperproperties = sets of *sets of traces*

Beyond Properties: Hyperproperties

- Properties = sets of traces
- capture a single run (the trace) of any program (the set)

  Hyperproperties = $\mathbf{sets}$ of $\mathit{sets\ of\ traces}$
- capture multiple runs (the $\mathit{sets\ of\ traces}$) of any program (the $\mathbf{sets}$)

Example: NonInterference

- NI: two different high inputs result in the same low outputs

Example: NonInterference

- NI: two different high inputs result in the same low outputs
- high = secret, low = public

Example: NonInterference

- NI: two different high inputs result in the same low outputs
- high = secret, low = public
- a set of traces tells all the behaviours of the same program with different high inputs

NI :

$$\left\{ \{t_1, t_2\} \;\middle|\; \begin{array}{l} \forall t_1, t_2 \in \{t_1, t_2\}. \\ \quad \text{if } \mathtt{inputs}(t_1) =_L \mathtt{inputs}(t_2) \\ \quad \text{then } \mathtt{outputs}(t_1) =_L \mathtt{outputs}(t_2) \end{array} \right\}$$

Example: Average Response Time < 1

ART :

$$\left\{ \{t \cdots\} \;\middle|\; \texttt{mean}\left(\bigcup_{t \in \{t \cdots\}} \texttt{response\_time}\,(t) \right) < 1 \right\}$$

where $\texttt{response\_time}\,(\cdot)$ looks in trace t and checks time between $\texttt{req}\,(\cdot)$ and $\texttt{resp}\,(\cdot)$

Hypersafety and Hyperliveness

Like Properties, Hyperproperties are partitioned in

- Hypersafety: something bad does not happen (NI)

Hypersafety and Hyperliveness

Like Properties, Hyperproperties are partitioned in

- Hypersafety: something bad does not happen (NI)
- Hyperliveness: something good eventually happens (ART)

Hypersafety as a dual

- Take the sets of traces that define a hypersafety property

Hypersafety as a dual

- Take the sets of traces that define a hypersafety property
- Describe hypersafety by the so-called sets of sets of bad prefixes

Hypersafety as a dual

- Take the sets of traces that define a hypersafety property
- Describe hypersafety by the so-called sets of sets of bad prefixes

NI-dual :

$$\left\{ \{t_1, t_2\} \;\middle|\; \begin{array}{l} \forall t_1, t_2 \in \{t_1, t_2\}. \\ \quad \text{if } \mathtt{inputs}(t_1) =_L \mathtt{inputs}(t_2) \\ \quad\; \text{then } \mathtt{outputs}(t_1) \neq_L \mathtt{outputs}(t_2) \end{array} \right\}$$

Property Satisfaction

How do we formalise a program having a property?

Property Satisfaction

How do we formalise a program having a property?

- P generates trace t: $P \rightsquigarrow t$

Property Satisfaction

How do we formalise a program having a property?

- P generates trace t: $P \rightsquigarrow t$
- Property $\pi = \{t\}$

Property Satisfaction

How do we formalise a program having a property?

- P generates trace t: $P \rightsquigarrow t$
- Property $\pi = \{t\}$
- $\vdash P : \pi \stackrel{\text{def}}{=}$ if $P \rightsquigarrow t$ then $t \in \pi$

Hyperproperty Satisfaction

How do we formalise a program having a hyperproperty?

Hyperproperty Satisfaction

How do we formalise a program having a hyperproperty?

- All traces generated by P:
  $\texttt{Behav}\,(P) = \{t \mid P \rightsquigarrow t\}$.

Hyperproperty Satisfaction

How do we formalise a program having a hyperproperty?

- All traces generated by P: $\texttt{Behav}(P) = \{t \mid P \rightsquigarrow t\}$.
- Hyperproperty $H = \{\{t\}\}$

Hyperproperty Satisfaction

How do we formalise a program having a hyperproperty?

- All traces generated by P: $\texttt{Behav}(P) = \{t \mid P \rightsquigarrow t\}$.
- Hyperproperty $H = \{\{t\}\}$
- $\vdash P : H \stackrel{\text{def}}{=} \texttt{Behav}(P) \in H$

Robustness

(Hyper)Properties must hold robustly:

- property satisfaction for whole programs protects against our bugs

Robustness

(Hyper)Properties must hold robustly:

- property satisfaction for whole programs protects against our bugs
- robust property satisfaction protects against any active adversary

Robustness

(Hyper)Properties must hold robustly:

- property satisfaction for whole programs protects against our bugs
- robust property satisfaction protects against any active adversary

So we want our program P to satisfy NRW, GS, NI or ART: $\forall \mathfrak{C}.\mathfrak{C}[P]$, so $\Theta = \mathfrak{C}[P]$

Robustness

(Hyper)Properties must hold robustly:

- property satisfaction for whole programs protects against our bugs
- robust property satisfaction protects against any active adversary

So we want our program P to satisfy NRW, GS, NI or ART: $\forall \mathfrak{C}. \mathfrak{C}[P]$, so $\Theta = \mathfrak{C}[P]$

Reminiscent of contextual equivalence!

Robust (Hyper)Property Satisfaction

How do we formalise a program having a hyperproperty robustly?

Robust (Hyper)Property Satisfaction

How do we formalise a program having a hyperproperty robustly?

- P now is a partial program

Robust (Hyper)Property Satisfaction

How do we formalise a program having a hyperproperty robustly?

- P now is a partial program
- $\mathfrak{C}$ is what P is linked against

Robust (Hyper)Property Satisfaction

How do we formalise a program having a hyperproperty robustly?

- P now is a partial program
- $\mathfrak{C}$ is what P is linked against
- $\vdash_R P : \pi \overset{\text{def}}{=} \forall \mathfrak{C}.$ if $\mathfrak{C}[P] \rightsquigarrow t$ then $t \in \pi$

Robust (Hyper)Property Satisfaction

How do we formalise a program having a hyperproperty robustly?

- P now is a partial program
- $\mathfrak{C}$ is what P is linked against
- $\vdash_R P : \pi \stackrel{\text{def}}{=} \forall \mathfrak{C}.$ if $\mathfrak{C}[P] \rightsquigarrow t$ then $t \in \pi$
- $\vdash_R P : H \stackrel{\text{def}}{=} \forall \mathfrak{C}.\mathtt{Behav}(\mathfrak{C}[P]) \in H$

A Note on Robustness

- Contexts can generate property-relevant events now

A Note on Robustness

- Contexts can generate property-relevant events now
- so they can trivially invalidate any property

A Note on Robustness

- Contexts can generate property-relevant events now
- so they can trivially invalidate any property
- we must filter events and consider only those generated by P

Example: Robust Safety

- $\pi \in Safety$
- $\vdash_R P : \pi \stackrel{\text{def}}{=} \forall \mathfrak{C}.$ if $\mathfrak{C}[P] \rightsquigarrow t$ then $t \in \pi$

Example: Robust Safety

- $\pi \in \mathit{Safety}$
- $\vdash_R P : \pi \stackrel{\text{def}}{=} \forall \mathfrak{C}.$ if $\mathfrak{C}[P] \rightsquigarrow t$ then $t \in \pi$
- dually: $\{m\} :: \pi \in \mathit{Safety}$
- $m \leq t$ = m is a prefix of t
- $\vdash_R P : \{m\} \stackrel{\text{def}}{=} \forall \mathfrak{C}.$ if $\mathfrak{C}[P] \rightsquigarrow t$ then $\nexists\, m \in \{m\}. m \leq t$

Example: Robust Liveness . . . ?

- can this hold robustly?

Example: Robust Liveness ...?

- can this hold robustly?
- we need a fair context in our setup: a context that will interact with us

Example: Robust Liveness . . . ?

- can this hold robustly?
- we need a fair context in our setup: a context that will interact with us
- avoid DOS: the attacker wants to violate our code, not starve it

Robust Compilation

1. specify (hyper)properties on programs through traces

Robust Compilation

1. specify (hyper)properties on programs through traces
2. specify (hyper)properties robustly

Robust Compilation

1. specify (hyper)properties on programs through traces
2. specify (hyper)properties robustly

Q: can we preserve them through compilation?

Robust Compilation

1. specify (hyper)properties on programs through traces
2. specify (hyper)properties robustly

Q: can we preserve them through compilation?

Yes!

Assumptions

- same alphabet of traces between S and T (I/O or syscalls)
- we lift this (partially) later

Example: Robust Property Preservation

- Assume the source has a property robustly

Example: Robust Property Preservation

- Assume the source has a property robustly
- Prove the compiled program has the same property robustly

Example: Robust Property Preservation

- Assume the source has a property robustly
- Prove the compiled program has the same property robustly

$$RTP : \forall \pi.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

Example: Robust Safety Preservation

- Same as RTP, restrict to safety

Example: Robust Safety Preservation

- Same as RTP, restrict to safety

$$
\begin{aligned}
RSP : & \forall \pi \in Safety.\ \forall \mathsf{P}. \\
& (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow \\
& (\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)
\end{aligned}
$$

Evaluation

Correct definitions

Evaluation

Correct definitions

Hard to use: no proof support

Evaluation

Correct definitions

Hard to use: no proof support

We want equivalent criteria that are easy to prove

Example: Robust Property Preservation #2

$$RTP : \forall \pi.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

Example: Robust Property Preservation #2

$$RTP: \forall \pi.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

$$\Updownarrow$$

Example: Robust Property Preservation #2

$$RTP : \forall \pi.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

$$\Updownarrow$$

$$PFRTP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall t.\ \ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow$$
$$\exists\ \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

RTP Intuition

If any trace in the target is also done in the source, and the source has the property, so does the target.

Example: Robust Safety Preservation #2

$$RSP: \forall \pi \in Safety.\ \forall \mathsf{P}.$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

Example: Robust Safety Preservation #2

$$RSP : \forall \pi \in Safety.\ \forall \mathsf{P}.$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow$$
$$(\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)$$

$$\Updownarrow$$

Example: Robust Safety Preservation #2

$$
\begin{aligned}
RSP : & \forall \pi \in Safety.\ \forall \mathsf{P}. \\
& (\forall \mathfrak{C}\ t.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t \Rightarrow t \in \pi) \Rightarrow \\
& (\forall \mathfrak{C}\ t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow t \in \pi)
\end{aligned}
$$

$$\Updownarrow$$

$$
\begin{aligned}
PFRSP : & \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall m. \\
& \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow m \Rightarrow \\
& \exists \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow m
\end{aligned}
$$

RSP Intuition

Safety is defined dually as a set of bad prefixes

If any prefix done in the target is also done in the source and the source has the safety property, that prefix is not bad, so the target also has the safety property

Relating RTP and RSP

- $RTP \iff PFRTP$

Relating RTP and RSP

- $RTP \iff PFRTP$
- $RSP \iff PFRSP$

Relating RTP and RSP

- $RTP \iff PFRTP$
- $RSP \iff PFRSP$
- $RTP \Rightarrow RSP$

Relating RTP and RSP

- $RTP \iff PFRTP$
- $RSP \iff PFRSP$
- $RTP \Rightarrow RSP$

Example: Robust HP Preservation

- as before: Assume the source has a hyperproperty robustly

Example: Robust HP Preservation

- as before: Assume the source has a hyperproperty robustly
- Prove the compiled program has the same hyperproperty robustly

Example: Robust HP Preservation

- as before: Assume the source has a hyperproperty robustly
- Prove the compiled program has the same hyperproperty robustly

$$RHP : \forall H.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}.\ \mathtt{Behav}\,(\mathfrak{C}[\mathsf{P}]) \in H) \Rightarrow$$
$$(\forall \mathfrak{C}.\ \mathtt{Behav}\,(\mathfrak{C}[[\![\mathsf{P}]\!]]) \in H)$$

Example: Robust HP Preservation #2

$$RHP : \forall H.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}.\ \texttt{Behav}(\mathfrak{C}[\mathsf{P}]) \in H) \Rightarrow (\forall \mathfrak{C}.\ \texttt{Behav}(\mathfrak{C}[[\![\mathsf{P}]\!]]) \in H)$$

Example: Robust HP Preservation #2

$$RHP : \forall H.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}.\ \mathtt{Behav}(\mathfrak{C}[\mathsf{P}]) \in H) \Rightarrow$$
$$(\forall \mathfrak{C}.\ \mathtt{Behav}(\mathfrak{C}[[\![\mathsf{P}]\!]]) \in H)$$

$$\Updownarrow$$

Example: Robust HP Preservation #2

$$RHP : \forall H.\ \forall \mathsf{P}.\ (\forall \mathfrak{C}.\ \mathtt{Behav}(\mathfrak{C}[\mathsf{P}]) \in H) \Rightarrow (\forall \mathfrak{C}.\ \mathtt{Behav}(\mathfrak{C}[[\![\mathsf{P}]\!]]) \in H)$$

$$\Updownarrow$$

$$PFRHP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \exists \mathfrak{C}.\ \mathtt{Behav}(\mathfrak{C}[[\![\mathsf{P}]\!]]) = \mathtt{Behav}(\mathfrak{C}[\mathsf{P}])$$

$$PFRHP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \exists \mathfrak{C}.\ \forall t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \iff \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

Quiz: Spot the Differences

$$PFRTP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall t.\ \ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \Rightarrow \exists\ \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

$$PFRHP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \exists \mathfrak{C}.\ \forall t.\ \mathfrak{C}[[\![\mathsf{P}]\!]] \rightsquigarrow t \iff \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

Answers

- Quantifier ordering

Answers

- Quantifier ordering
- Implication

Answers

- Quantifier ordering
- Implication

Intuition

- Quantifier ordering: lifts to sets of traces since a $\mathfrak{C}$ in PFRHP works for a set of traces

Answers

- Quantifier ordering
- Implication

Intuition

- Quantifier ordering: lifts to sets of traces since a $\mathfrak{C}$ in PFRHP works for a set of traces
- Implication: a single implication means refinement, so the target can have more behaviours.

Answers

- Quantifier ordering
- Implication

Intuition

- Quantifier ordering: lifts to sets of traces since a $\mathfrak{C}$ in PFRHP works for a set of traces
- Implication: a single implication means refinement, so the target can have more behaviours. Co-implicaiton means no refinement, we need the exact same traces to ensure inclusion in the H

Example: Robust Hypersafety Preservation

$$PFRHSP: \forall \mathsf{P}. \forall \mathfrak{C}. \forall \{m\}.$$
$$\{m\} \leq \texttt{Behav}(\mathfrak{C}[[\![\mathsf{P}]\!]]) \Rightarrow \exists \mathfrak{C}. \{m\} \leq \texttt{Behav}(\mathfrak{C}[\mathsf{P}])$$

Example: Robust Hypersafety Preservation

$$PFRHSP: \ \forall \mathsf{P}. \ \forall \mathfrak{C}. \ \forall \{m\}.$$
$$\{m\} \leq \mathtt{Behav}\,(\mathfrak{C}[\llbracket \mathsf{P} \rrbracket]) \Rightarrow \exists \mathfrak{C}. \ \{m\} \leq \mathtt{Behav}\,(\mathfrak{C}[\mathsf{P}])$$

Where $\leq$ means *all* prefixes of $\{m\}$ are extended by the behaviour of the (compiled) program

Subclasses of Hyperproperties

- K-Hypersafety: hypersafety for sets of cardinality k (if $k = 4$, NMIF)

Subclasses of Hyperproperties

- K-Hypersafety: hypersafety for sets of cardinality k (if $k = 4$, NMIF)
- 2-Hypersafety: hypersafety for sets of cardinality 2: set of pairs of traces: NI

Subclasses of Hyperproperties

- K-Hypersafety: hypersafety for sets of cardinality k (if $k = 4$, NMIF)
- 2-Hypersafety: hypersafety for sets of cardinality 2: set of pairs of traces: NI
- Subset-closed HP: set of traces closed under subsetting

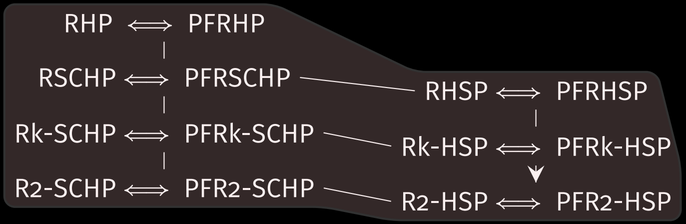
Subclasses of Hyperproperties

- K-Hypersafety: hypersafety for sets of cardinality k (if $k = 4$, NMIF)
- 2-Hypersafety: hypersafety for sets of cardinality 2: set of pairs of traces: NI
- Subset-closed HP: set of traces closed under subsetting
- K-, 2- Subset-closed HP: as before, curtail set cardinality to k, 2

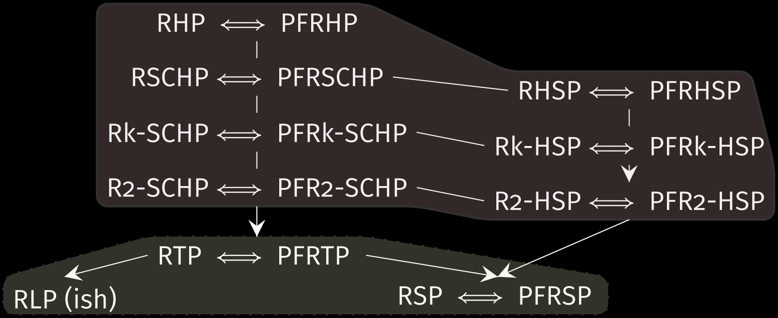
Subclasses of Hyperproperties

- K-Hypersafety: hypersafety for sets of cardinality k (if $k = 4$, NMIF)
- 2-Hypersafety: hypersafety for sets of cardinality 2: set of pairs of traces: NI
- Subset-closed HP: set of traces closed under subsetting
- K-, 2- Subset-closed HP: as before, curtail set cardinality to k, 2
- Hyperliveness: not present: RHLP collapses with RHP

Robust Compilation (RC) Diagram



Robust Compilation (RC) Diagram



RC vs FAC

- (some) RC criteria are propositional

RC vs FAC

- (some) RC criteria are propositional
  (some are relational but they are not presented here)
- FAC is only relational

RC vs FAC

- (some) RC criteria are propositional
  (some are relational but they are not presented here)
- FAC is only relational
- both are robust

RC vs FAC

- (some) RC criteria are propositional
  (some are relational but they are not presented here)
- FAC is only relational
- both are robust
- FAC is only as precise as the equivalence
- RC do not preserve abstractions beyond the related security (hyper)property

Proving RC

$$PFRTP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall t.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow t \Rightarrow \exists\ \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

$$PFRSP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall m.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow m \Rightarrow \exists \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow m$$

Proving RC

$$PFRTP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall t.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow t \Rightarrow \exists\ \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

$$PFRSP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall m.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow m \Rightarrow \exists \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow m$$

Recall $\Rightarrow$ for FAC (contrapositive):

Proving RC

$$PFRTP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall t.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow t \Rightarrow \exists\ \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow t$$

$$PFRSP : \forall \mathsf{P}.\ \forall \mathfrak{C}.\ \forall m.$$

$$\mathfrak{C}[\llbracket \mathsf{P} \rrbracket] \rightsquigarrow m \Rightarrow \exists \mathfrak{C}.\ \mathfrak{C}[\mathsf{P}] \rightsquigarrow m$$

Recall $\Rightarrow$ for FAC (contrapositive):

$\forall \mathsf{P}_1, \mathsf{P}_2$

$$\exists \mathfrak{C}.\mathfrak{C}[\llbracket \mathsf{P}_1 \rrbracket]\Uparrow \not\Leftrightarrow \mathfrak{C}[\llbracket \mathsf{P}_2 \rrbracket] \Rightarrow \exists \mathfrak{C}.\mathfrak{C}[\mathsf{P}_1]\Uparrow \not\Leftrightarrow \mathfrak{C}[\mathsf{P}_2]\Uparrow$$

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have
- $\mathfrak{C}$, t for PFRTP

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have
- $\mathfrak{C}$, t for PFRTP
- $\mathfrak{C}$, m for PFRSP

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have
- $\mathfrak{C}$, t for PFRTP
- $\mathfrak{C}$, m for PFRSP
- $\mathfrak{C}$, only!! for PFRHP

Backtranslation!

- generate a $\mathfrak{C}$ starting from what we have
- $\mathfrak{C}$, t for PFRTP
- $\mathfrak{C}$, m for PFRSP
- $\mathfrak{C}$, only!! for PFRHP
- $\mathfrak{C}$, {m} for PFRHSP

Backtranslation!

- $m / \{m\}$ yields trace-based BT

Backtranslation!

- $m/\{m\}$ yields trace-based BT

Backtranslation!

- $m/\{m\}$ yields trace-based BT
- t is infinite, $\mathfrak{C}$ is finite, so only use $\mathfrak{C}$ there

Backtranslation!

- $m / \{m\}$ yields trace-based BT
- t is infinite, $\mathfrak{C}$ is finite, so only use $\mathfrak{C}$ there
- $\mathfrak{C}$ yields context-based BT

Backtranslation!

- $m/\{m\}$ yields trace-based BT
- t is infinite, $\mathfrak{C}$ is finite, so only use $\mathfrak{C}$ there
- $\mathfrak{C}$ yields context-based BT
    - can be precise BT

Backtranslation!

- $m/\{m\}$ yields trace-based BT
- t is infinite, $\mathfrak{C}$ is finite, so only use $\mathfrak{C}$ there
- $\mathfrak{C}$ yields context-based BT
  - can be precise BT
  - or approximate BT (intuitively analogous to trace-based BT)

Backtranslation!

- $m / \{m\}$ yields trace-based BT
- t is infinite, $\mathfrak{C}$ is finite, so only use $\mathfrak{C}$ there
- $\mathfrak{C}$ yields context-based BT
  - can be precise BT
  - or approximate BT (intuitively analogous to trace-based BT)
- BT is not the inverse of compilation

Conclusion

We have seen:

- Properties and Hyperproperties: to formalise a program having a securty property
- Robust compilation criteria, which preserve classes of (hyper)properties
- Backtranslation-equivalent Robust compilation criteria