SecRGSep

Combining Information Flow Security and RGSep

Vincent Jackson[1,2] Toby Murray[1], Christine Rizkallah[1]

The University Of Melbourne[1] and UNSW Sydney[2]

June 4, 2026

Introduction: Motivation

Problem Area: Proof Methods for Concurrent Information Flow Security

1. Concurrent programs are hard to reason about.
2. Information Flow is hard to reason about.

Introduction: Summary

Approach: Relational Logic + RGSep

Results:

- SecRGSep: Relational Info-flow logic + RGSep
- Declassification as relational await
- General conditions for security of atomic update relations (specialise to whatever you want)
- The logic allows for observations of the local state
- Security proof proven directly in the relational semantics
- Results proven in Isabelle/HOL

Outline

- Introduction

  — we are here —
- Information Flow Security
  - Two-run Observations
  - Reasoning About Information Flow Security
- SecRGSep
  - Components of Security
    - Semantic Security
    - Branch Security
  - Separation Logic & Information Flow Security
    - RGSep Background
    - Separation Logic & Information Flow Security
    - New Rules
    - Theorem: Proof Lifting
    - Theorem: SecRGSep Security
- Conclusion

Information Flow Security

What is Information Flow Security?

Confidentiality: Secret inputs are not revealed via non-secret outputs.
Integrity: Untrusted inputs do not affect trusted outputs.

Same form, different labels.
I focus on confidentiality.

Why is Information Flow Security Important?

Secrets: passwords, health data, classified documents.

Low-High Variables

$\mathbf{var}\, h, h_2\ \mathbf{is}$ high

$\mathbf{var}\, l\ \mathbf{is}$ low

$h := h_2$ ✓

$l := h$ ✗

Low-High Variables

$\mathbf{var}\, h, h_2\ \mathbf{is}$ high

$\mathbf{var}\, l\ \mathbf{is}$ low

$h := h_2$ ✓

$l := h$ ✗

Output Statements

$h := h_2$

$\langle l := h; \mathbf{output}\, h \rangle$

Information Flow Security: Two-run Observations

A program is *information flow secure* if (starting from indistinguishable states) *every* execution has the *same* observable behaviour.

Contraposing, we obtain:

Any *pair* of executions (starting in indistinguishable states) that have *different* observable behaviours shows the program is information flow *insecure*.

(Information flow security is a Hyperproperty)

Information Flow Security: Two-run Observations

$$1, c_1 \longrightarrow 2, c_2 \longrightarrow 4, c_3 \longrightarrow 5, c_4$$

$$3, c_1 \longrightarrow 4, c_2 \longrightarrow 6, c_3 \longrightarrow 6, c_4$$

Information Flow Security: Two-run Observations

This example: compare runs using $obs = \lambda x.\ x\%2$

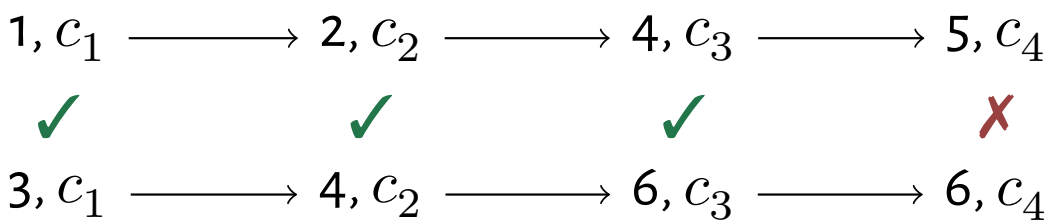
$$1, c_1 \longrightarrow 2, c_2 \longrightarrow 4, c_3 \longrightarrow 5, c_4$$

✓

$$3, c_1 \longrightarrow 4, c_2 \longrightarrow 6, c_3 \longrightarrow 6, c_4$$

Information Flow Security: Two-run Observations

This example: compare runs using $obs = \lambda x.\ x\%2$



Information Flow Security: Two-run Observations

This example: compare runs using $obs = \lambda x.\ x\%2$

Altenative mismatch: different programs.

$$
\begin{array}{ccccccc}
1, c_1 & \longrightarrow & 2, c_2 & \longrightarrow & 4, c_3 & \longrightarrow & 5, c_4 \\
\checkmark & & \checkmark & & \checkmark & & \times \\
3, c_1 & \longrightarrow & 4, c_2 & \longrightarrow & 6, c_3 & \longrightarrow & 7, c_4'
\end{array}
$$

Reasoning about Information Flow Security

A standard method is to use *relational logic*.
Assertions have the type $\text{State} \times \text{State} \to \text{bool}$.

Standard boolean operations: $\wedge, \vee, \Rightarrow, \neg$.

New Things:

Predicate Lifting: $\langle\!| p_1 \mid p_2 |\!\rangle \;\widehat{=}\; \lambda(x, y).\ p_1\, x \wedge p_2\, y$
(when $p_1 = p_2 = p$, write $\langle\!| p |\!\rangle$)

Agreement (for observations): $\mathbb{A}\, obs \;\widehat{=}\; \lambda(x, y).\ obs\, x = obs\, y$
(where obs is a function)

Reasoning about Information Flow Security

A standard method is to use *relational logic*.
Assertions have the type $\text{State} \times \text{State} \to \text{bool}$.

Agreement (for observations): $\mathbb{A}\, obs \;\widehat{=}\; \lambda(x, y).\; obs\, x = obs\, y$

$$\begin{array}{ll} \{p \wedge \mathbb{A}\, obs\} & \{p\} \\ \textbf{output}\ obs & \textbf{declassify}\ obs \\ \{p\} & \{p \wedge \mathbb{A}\, obs\} \end{array}$$

Declassification

Observation

Declassify: Emit a value. The attacker *learns* this value.

Compare:

$$\{p\}$$
$$\textbf{declassify}\ obs$$
$$\{p \wedge \mathbb{A}\, obs\}$$

$$\{p\}$$
$$\textbf{await}\ q$$
$$\{p \wedge q\}$$

Declassify = Await on Agreement.

$$\{p\}$$
$$\textbf{await}\ (\mathbb{A}\, obs)$$
$$\{p \wedge \mathbb{A}\, obs\}$$

Outline

- Introduction
- Information Flow Security
  - Two-run Observations
  - Reasoning About Information Flow Security

  — we are here —
- SecRGSep
  - Components of Security
    - Semantic Security
    - Branch Security
  - Separation Logic & Information Flow Security
    - RGSep Background
    - Separation Logic & Information Flow Security
    - New Rules
    - Theorem: Proof Lifting
    - Theorem: SecRGSep Security
- Conclusion

SecRGSep

SecRGSep: Information Flow Security RGSep

Relational information flow logic + RGSep

(RGSep = rely–guarantee + separation logic).

GenRGSep: General RGSep implementation [Jackson et al., 2024]

(in Isabelle/HOL)

Main aim: Relational Information-Flow Reasoning + GenRGSep

SecRGSep: Two-state Executions

Natural semantics of relational logic:

$$\begin{pmatrix}1\\3\end{pmatrix}, c_1 \longrightarrow \begin{pmatrix}2\\4\end{pmatrix}, c_2 \longrightarrow \begin{pmatrix}4\\6\end{pmatrix}, c_3 \longrightarrow \begin{pmatrix}5\\6\end{pmatrix}, c_4$$

The two components of SecRGSep security:

- Secure Atoms
  No atomic command illicitly leaks information
- Secure Branching
  The program always steps to the same final command from indistinguishable states

What Assertions are Information Flow Assertions?

Clearly good: $\langle\!|p|\!\rangle$, $\mathbb{A}\, obs$

Earlier work uses *partial equivalence relations* [Sabelfeld and Sands, 1999]: quasi-reflexive, symmetric, transitive.

Problem: *transitivity* is not preserved by *disjunction* ($\vee$) or *separating conjunction* ($*/\ *_{\wedge}$).

What Assertions are Information Flow Assertions?

Transitivity is not preserved by *disjunction* ($\vee$) or *separating conjunction* ($*/\ *_{\wedge}$).

What properties do we actually need?

Answer from *hyperset* semantics [Morgan, 2009].

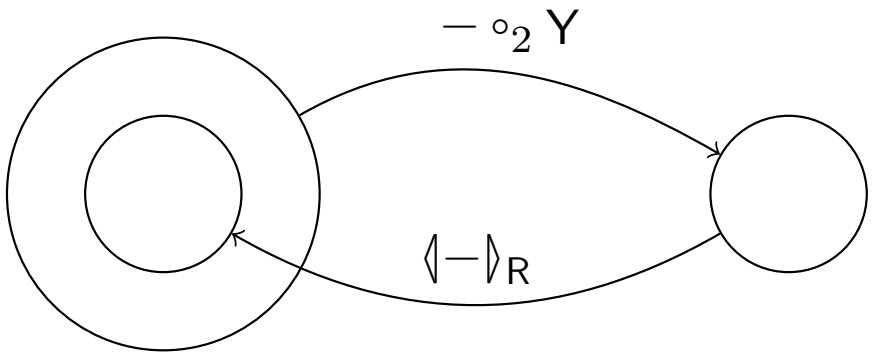
The *partial tolerance* relations: quasi-reflexivity and symmetry.

What Assertions are Information Flow Assertions?

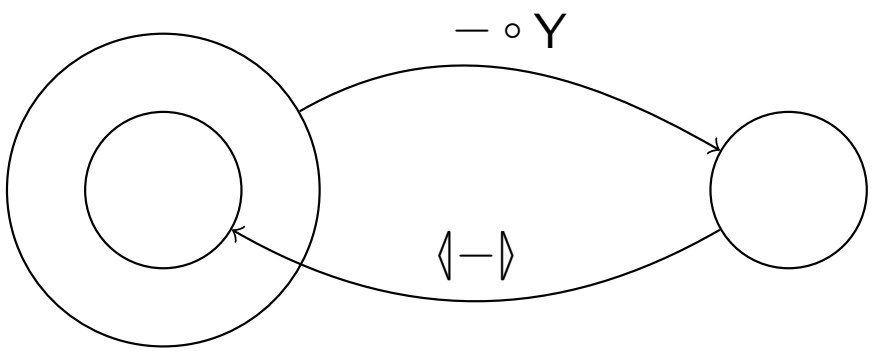
$\mathbb{A}\, obs$	$\mathrm{R}, \mathrm{Q}, \mathrm{S}, \mathrm{T}$
$\langle\!\langle p \rangle\!\rangle$	$\mathrm{Q}, \mathrm{S}, \mathrm{T}$
$p \wedge q$	$\mathrm{RR/R}, \mathrm{QQ/Q}, \mathrm{SS/S}, \mathrm{TT/T}$
$p \vee q$	$\top\mathrm{R/R}, \mathrm{R}\top\mathrm{/R}, \mathrm{QQ/Q}, \mathrm{SS/S}$
$p \Rightarrow q$	$\top\mathrm{R/R}, \mathrm{SS/S}$
$\neg p$	$\mathrm{S/S}$
$p * q$	$\mathrm{QQ/Q}, \mathrm{SS/S}$
$p *_{\wedge} q$	$\mathrm{QQ/Q}, \mathrm{SS/S}$

Semantic Security

Double Relations Standard Relations

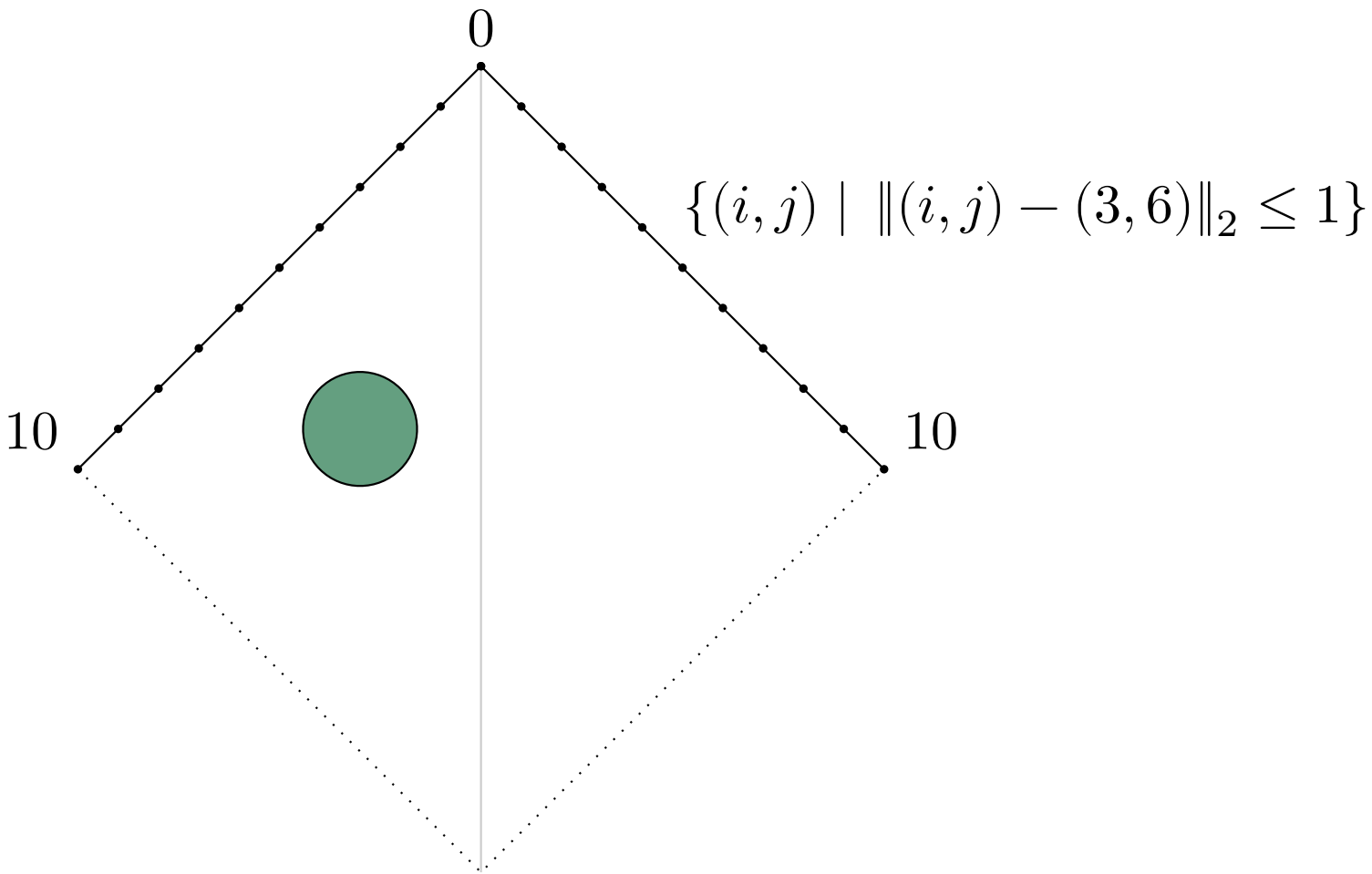


Double Predicates Standard Predicates



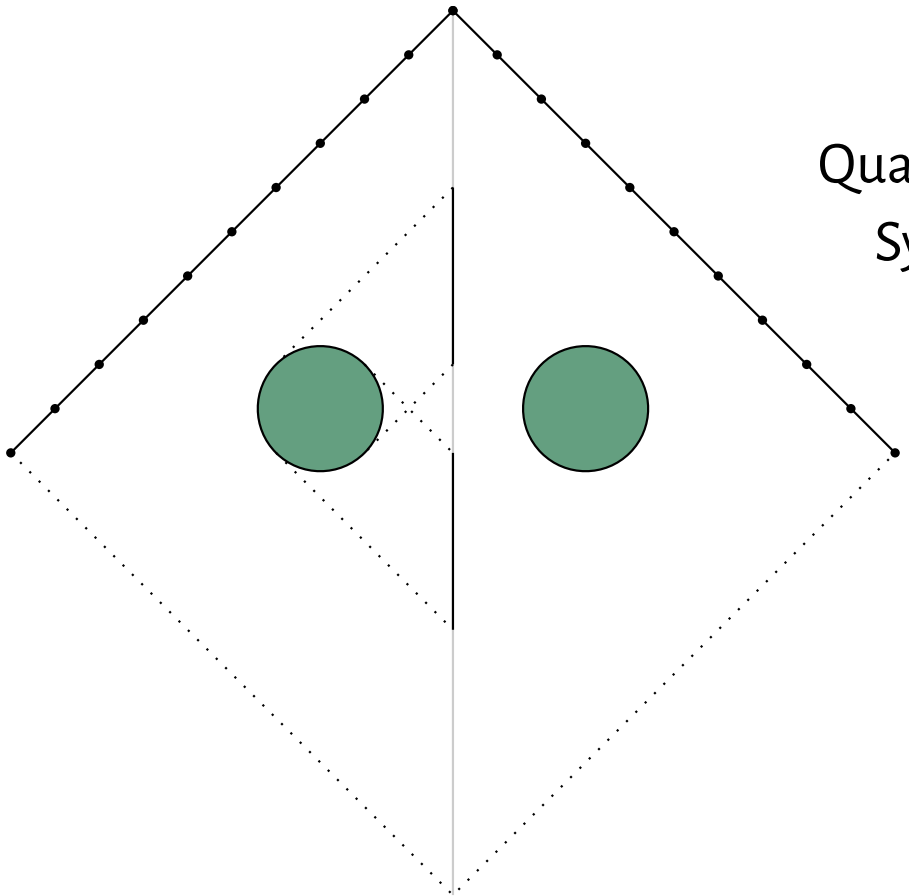
Semantic Security

$$\mathsf{Val} \times \mathsf{Val} \equiv [0, 10] \times [0, 10]$$



Semantic Security

$$\mathsf{Val} \times \mathsf{Val} \equiv [0, 10] \times [0, 10]$$

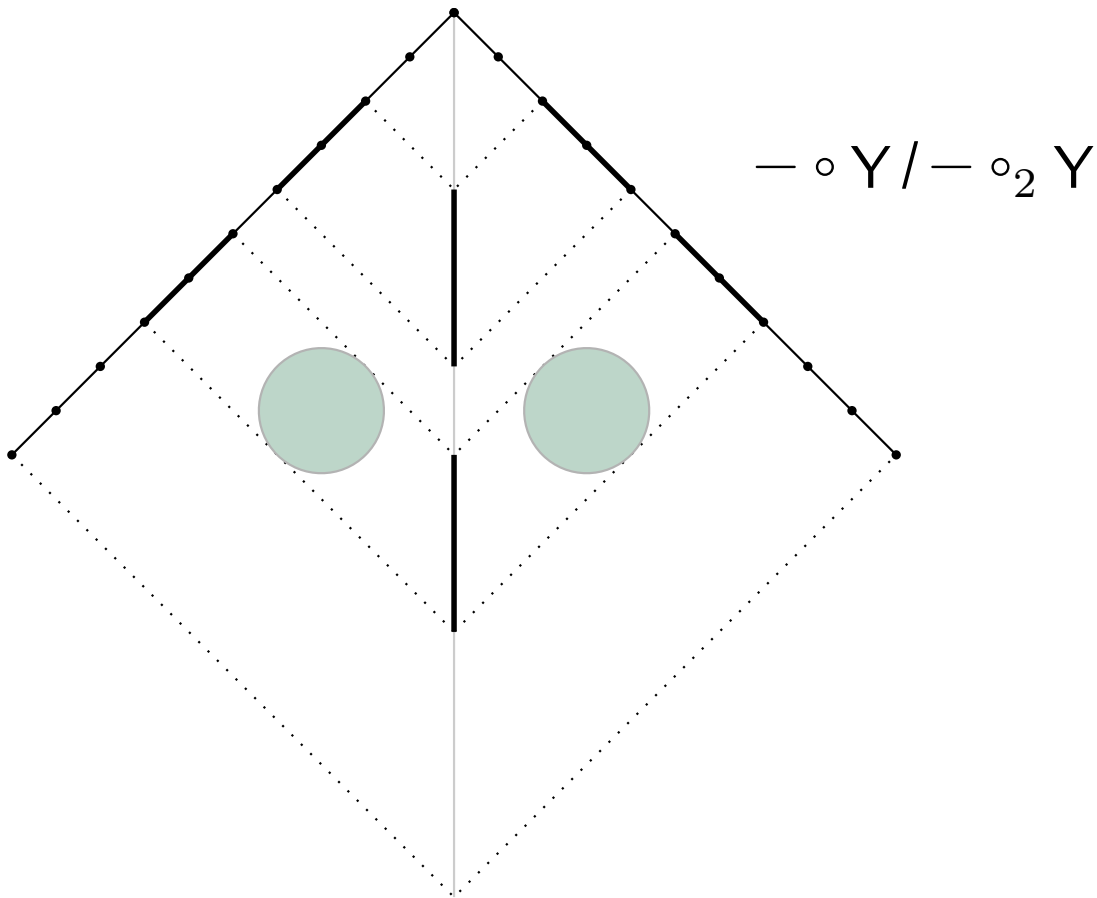


Quasi-reflexive Closed

Symmetry Closed

Semantic Security

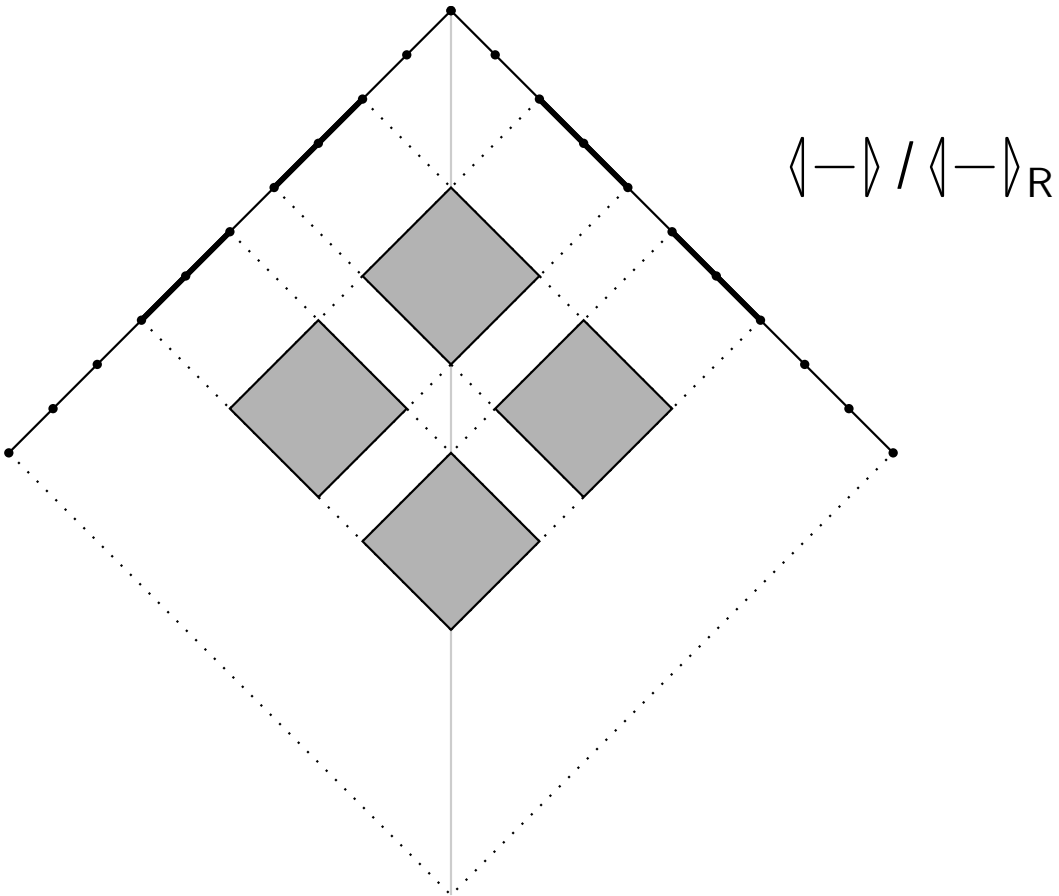
$$\mathsf{Val} \times \mathsf{Val} \equiv [0, 10] \times [0, 10]$$



$$- \circ \mathsf{Y} \,/\, - \circ_2 \mathsf{Y}$$

Semantic Security

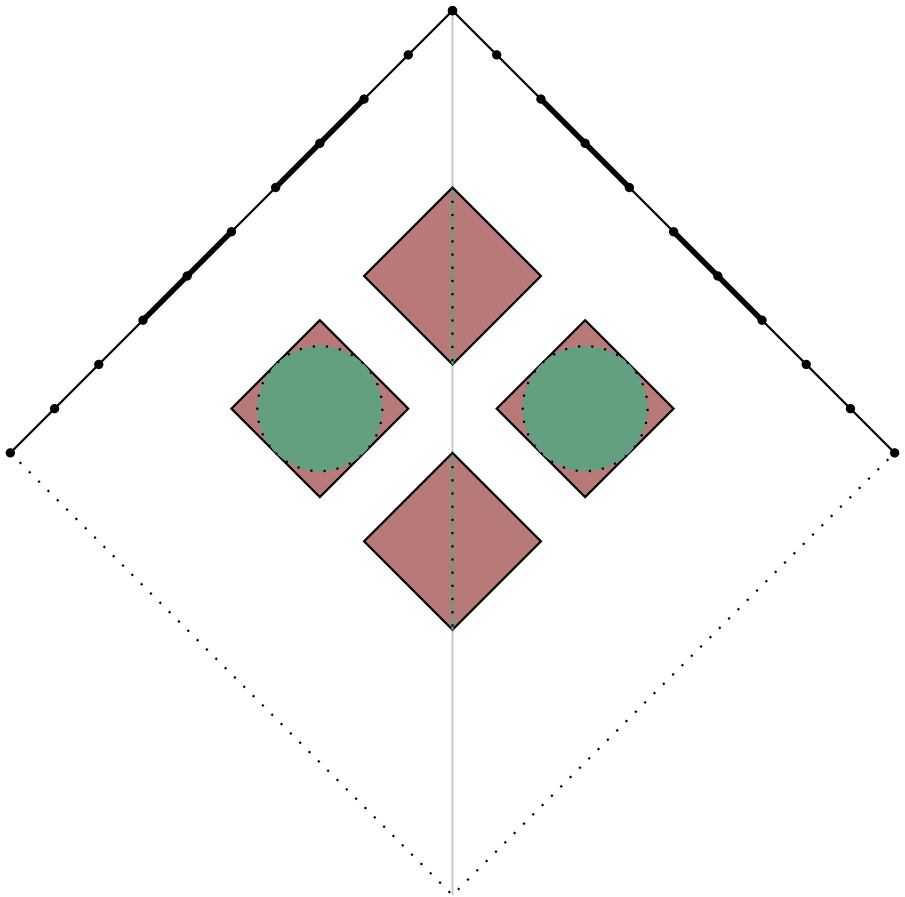
$$\mathsf{Val} \times \mathsf{Val} \equiv [0, 10] \times [0, 10]$$



$\langle\!|-|\!\rangle \,/\, \langle\!|-|\!\rangle_{\mathsf{R}}$

Semantic Security

$$\mathsf{Val} \times \mathsf{Val} \equiv [0, 10] \times [0, 10]$$



Healthiness and Information-Flow Properties

Healthiness Conditions

Quasi-reflexivity Preserving: $r \subseteq \langle r \circ_2 \mathsf{Y} \rangle_R$ ‘● ⊆ ◆’

Symmetry Preserving: $r = \langle r \circ_2 \mathsf{X} \rangle_R$

Information-Flow Conditions (◆ − ● = ◌)

Revealing: $\text{pre-state}\,(\langle r \circ_2 \mathsf{Y} \rangle_R - r)$

Declassifying: $\langle \text{pre-state}\, r \circ \mathsf{Y} \rangle - \text{pre-state}\, r$

Insecure Pre-states: $\text{Revealing} - \text{Declassifying}$

$\mathsf{Y}\, x = (x, x)$

$\mathsf{X}\,(x, y) = (y, x)$

Atomic Security

Non-Revealing: $\neg\,\text{pre-state}\,(\langle\!|r \circ_2 \mathsf{Y}|\!\rangle_{\mathsf{R}} \Rightarrow r)$

Declassifying: $\langle\!|\,\text{pre-state}\,r \circ \mathsf{Y}|\!\rangle - \text{pre-state}\,r$

Secure Pre-states: $\text{Non-revealing} \vee \text{Declassifying}$

$$\text{Non-revealing}\,\langle\!|r|\!\rangle_{\mathsf{R}} \equiv \top$$
$$\text{Declassifying}\,\langle\!|r|\!\rangle_{\mathsf{R}} \equiv \top$$

$$\text{Non-revealing}\,(\mathbf{output}\,obs) \equiv \mathbb{A}\,obs$$
$$\text{Declassifying}\,(\mathbf{output}\,obs) \equiv \bot$$

$$\text{Non-revealing}\,(\mathbf{declassify}\,obs) \equiv \mathbb{A}\,obs$$
$$\text{Declassifying}\,(\mathbf{declassify}\,obs) \equiv \neg\,\mathbb{A}\,obs$$

The Two Components of SecRGSep Security

- Secure Atoms ✓
  No atomic command illicitly leaks information
- Secure Branching
  The program always steps to the same final command from indistinguishable states

Branch Security

GenRGSep uses a CSP like language.

There are three forms of non-determinism:

1. External Non-determinism: $(c_1 \,\square\, c_2)$
2. Do loop: $\mathbf{do}\ c\ \mathbf{od}$
3. Internal Non-determinism: $(c_1 \sqcap c_2)$

They are secure when:

1. External Non-determinism: stable (no internal moves) +

$$\neg\langle\!|\, \mathrm{enabled}\, c_1 \circ \mathsf{Y} \mid \mathrm{enabled}\, c_2 \circ \mathsf{Y} \,|\!\rangle \wedge$$
$$\neg\langle\!|\, \mathrm{enabled}\, c_2 \circ \mathsf{Y} \mid \mathrm{enabled}\, c_1 \circ \mathsf{Y} \,|\!\rangle$$

2. Do loop: stable + $\mathbb{A}\,(\mathrm{blocked}\, c)$
3. Internal Non-determinism: Never!

Enforce this in the precondition of each such sub-program.

The Two Components of SecRGSep Security

- Secure Atoms ✓
  No atomic command illicitly leaks information
- Secure Branching ✓
  The program always steps to the same final command from indistinguishable states

Outline

- Introduction
- Information Flow Security
  - Two-run Observations
  - Reasoning About Information Flow Security
  - Password Example
- SecRGSep
  - Components of Security
    - Semantic Security
    - Branch Security

    — we are here —
  - Separation Logic & Information Flow Security
    - RGSep Background
    - Separation Logic & Information Flow Security
    - New Rules
    - Theorem: Proof Lifting
    - Theorem: SecRGSep Security
- Conclusion

RGSep Background

RGSep: Rely–Guarantee + Concurrent Separation Logic.

There are *two* parts to the state: *local* and *shared*.

Local state is *split* between each process (using $*$).

Shared state is *shared* between each process (using $\wedge$).

Combine predicates with $*_{\wedge}$.

The RGSep Judgement:

$$R, G \vdash \{ p \}\ c\ \{ q \}.$$

RGSep + Information-Flow Security

Problem 1:

For security, we need *invariant assertions*.

Thus: GenRGSep + Invariants.

Due to separation logic, we need *two* invariants.

The RGSep Judgement with invariants:

$$R, G, F, I \vdash \{ p \}\ c\ \{ q \}.$$

Problem 2:

(RGSep State) $(\texttt{'l} \times \texttt{'l}) \times (\texttt{'s} \times \texttt{'s}) \neq (\texttt{'l} \times \texttt{'s}) \times (\texttt{'l} \times \texttt{'s})$ (Relational State)

Exchange function $\ddagger((a, b), (c, d)) \mathrel{\widehat{=}} ((a, c), (b, d))$

Everything gets a ddagger subscript: $\mathbb{A}_{\ddagger}\ obs$, $\langle\!| a |\!\rangle_{\ddagger}$, $\text{secure-rel}_{\ddagger}$, etc.

Separation Logic & Information Flow Security

Enforce the security conditions by putting the invariants together:

$$
\begin{aligned}
R, G, I, F \vdash^{\mathsf{sec}} \{\, p \,\}\ c\ \{\, q \,\} \mathrel{\widehat{=}}\ & \\
& R, G, I, F \vdash \{\, p \,\}\ c\ \{\, q \,\} \wedge \\
& \left(I *_{\wedge} F \Rightarrow \text{secure-atoms}_{\ddagger}\ c\right) \wedge \\
& \left(I *_{\wedge} F \Rightarrow \text{secure-branching}_{\ddagger}\ c\right)
\end{aligned}
$$

SecRGSep: New Rules

$$\frac{\begin{array}{c} p \Rightarrow I \qquad q \Rightarrow I \\ p \textbf{ stable under } R \qquad q \textbf{ stable under } R \\ \forall f \Rightarrow F.\ \textbf{sp}\ a\ (p *_\wedge f) \Rightarrow q *_\wedge \text{any-shared}\ f \\ \text{snd}\ '_{\mathsf{R}}\ (\textbf{pretest}\ (p *_\wedge F) \wedge a) \Rightarrow G \\ \boxed{I *_\wedge F \Rightarrow \text{secure-prestate}_\ddagger\ a} \end{array}}{R, G, F, I \vdash^{\text{sec}} \{\, p \,\}\ \langle a \rangle\ \{\, q \,\}}\ \text{Atomic}$$

$$\frac{\begin{array}{c} R, G, F, I \vdash^{\text{sec}} \{\, i \,\}\ c\ \{\, i \,\} \\ \boxed{I *_\wedge F \Rightarrow \text{secure-branching}_\ddagger\ (\textbf{do}\ c\ \textbf{od})} \qquad i \textbf{ stable under } R \end{array}}{R, G, F, I \vdash^{\text{sec}} \{\, i \,\}\ \textbf{do}\ c\ \textbf{od}\ \{\, i \,\}}\ \text{Do}$$

$$\frac{\begin{array}{c} R, G, F, I \vdash^{\text{sec}} \{\, p \,\}\ c_1\ \{\, q_1 \,\} \\ R, G, F, I \vdash^{\text{sec}} \{\, p \,\}\ c_2\ \{\, q_2 \,\} \\ \boxed{I *_\wedge F \Rightarrow \text{secure-branching}_\ddagger\ (c_1 \,\square\, c_2)} \end{array}}{R, G, F, I \vdash^{\text{sec}} \{\, p \,\}\ c_1 \,\square\, c_2\ \{\, q_1 \vee q_2 \,\}}\ \text{ENDet}$$

The SecRGSep Logic: important rules. New side-conditions highlighted.

Information Flow Security: Proof Lifting

Theorem: Proof Lifting

$$\begin{aligned} \text{If} \quad & R, G, F, I \vdash_{\not\Downarrow} \{\, p \,\} \lfloor c \rfloor \{\, q \,\} \\ & \text{quasirefl-preserv}_{\ddagger}\, c \\ \text{then} \quad & \langle\!| R |\!\rangle_{\mathsf{R}}, \langle\!| G |\!\rangle_{\mathsf{R}}, \langle\!| F |\!\rangle_{\ddagger}, \langle\!| I |\!\rangle_{\ddagger} \vdash \{\, \langle\!| p |\!\rangle_{\ddagger} \,\}\, c \,\{\, \langle\!| q |\!\rangle_{\ddagger} \,\}\,. \end{aligned}$$

(This splitting is similar to the design of VERONICA [Schoepe et al., 2020].)

SecRGSep: Security Theorem

Theorem: Semantic Correctness Establishes Security

$$\text{If} \quad R, G, F, I \vdash^{\text{sec}} \{\, p \,\}\ c\ \{\, q \,\}$$
$$\text{then} \quad p \Rightarrow \text{secure-exec}\ R\, F\, c$$

$\text{secure-exec}\ R\, F\, c\, s$ means:
at every point in the execution of c from s (under the rely R and frame F),

1. The standard-state program produces the same post-program.
2. Every atomic command executes securely.

Conclusion

- SecRGSep: Relational Info-flow logic + RGSep
- Declassification as concurrent await
- General conditions for security of atomic update relations (specialise to whatever you want)
- The logic allows for observations of the local state
- Security proof proven directly in the relational semantics (two-run property emergent, rather than baked in)
- Results proven in Isabelle/HOL

Vincent Jackson, Toby Murray, and Christine Rizkallah. 2024. A Generalised Union of Rely-Guarantee and Separation Logic Using Permission Algebras. In *15th International Conference on Interactive Theorem Proving (ITP 2024) (Leibniz International Proceedings in Informatics (LIPIcs))*, Yves Bertot, Temur Kutsia, and Michael Norrish (Eds.), Vol. 309. Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl, Germany, 23:1–23:16. https://doi.org/10.4230/LIPIcs.ITP.2024.23

Carroll Morgan. 2009. How to Brew-up a Refinement Ordering. *Electronic Notes in Theoretical Computer Science* 259 (2009), 123–141. https://doi.org/10.1016/j.entcs.2009.12.021 Proceedings of the 14th BCS-FACS Refinement Workshop (REFINE 2009).

Andrei Sabelfeld and David Sands. 1999. A Per Model of Secure Information Flow in Sequential Programs. In *Programming Languages and Systems*, S. Doaitse Swierstra (Ed.). Springer Berlin Heidelberg, Berlin, Heidelberg, 40–58. https://doi.org/10.1023/A:1011553200337

Daniel Schoepe, Toby Murray, and Andrei Sabelfeld. 2020. VERONICA: Expressive and Precise Concurrent Information Flow Security. In *2020 IEEE 33rd Computer Security Foundations Symposium (CSF)*. 79–94. https://doi.org/10.1109/CSF49147.2020.00014

SecRGSep: Security Theorem

Proof Sketch:

1. Lemma 1: secure-branching implies control flow always matches.
2. Lemma 2: matched control flow means that a property true for every atomic *command* lifts to every atomic *step* of that program.[1]
3. atom-secure ensures every atomic command is secure.
4. Thus: SecRGSep ensures security.

1, Caveat: that behave nicely with loops & blocking.

SecRGSep: Healthiness Conditions

Matched control flow means that a property true for every atomic *command* lifts to every atomic *step* of that program.[1]

1, Caveat: that behave nicely with loops & blocking.

quasirefl-preserv is only nice if *no declassification occurs in loop guards*.