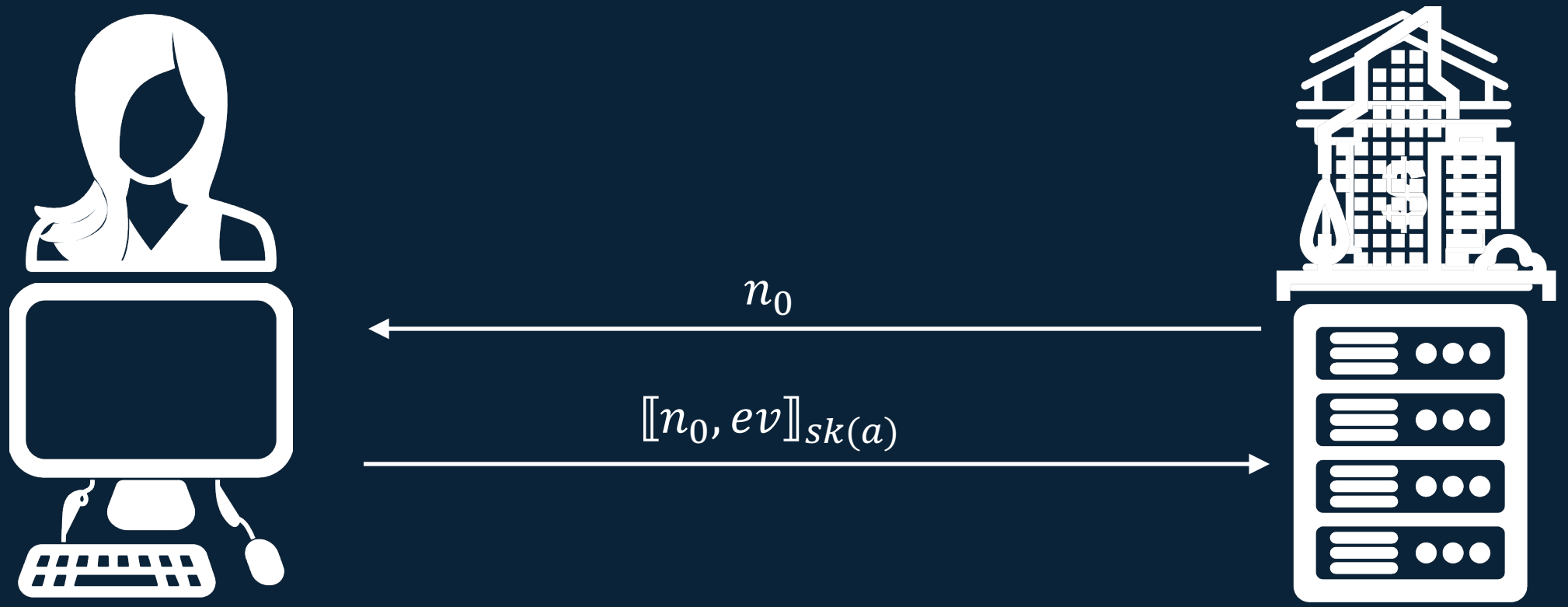
Automated Trust Analysis of Copland Specifications for Layered Attestation

Paul D. Rowe, John D. Ramsdell, and Ian D. Kretz

PPDP 2021
September 8, 2021

MITRE | SOLVING PROBLEMS FOR A SAFER WORLD

Remote Attestation

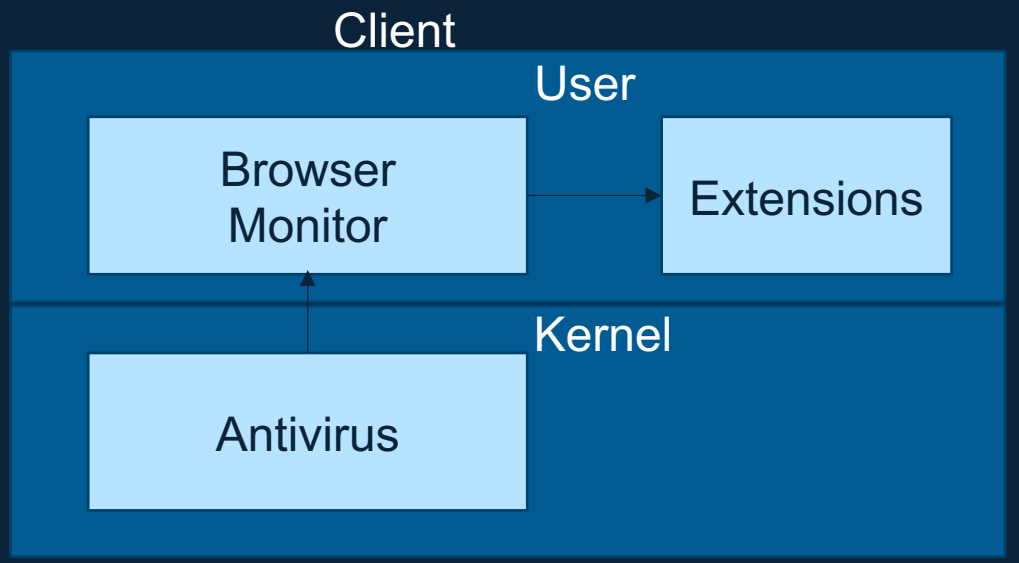


Bank: ev is evidence of no “man in the browser”

Office: ev is evidence of conformance to corporate configuration

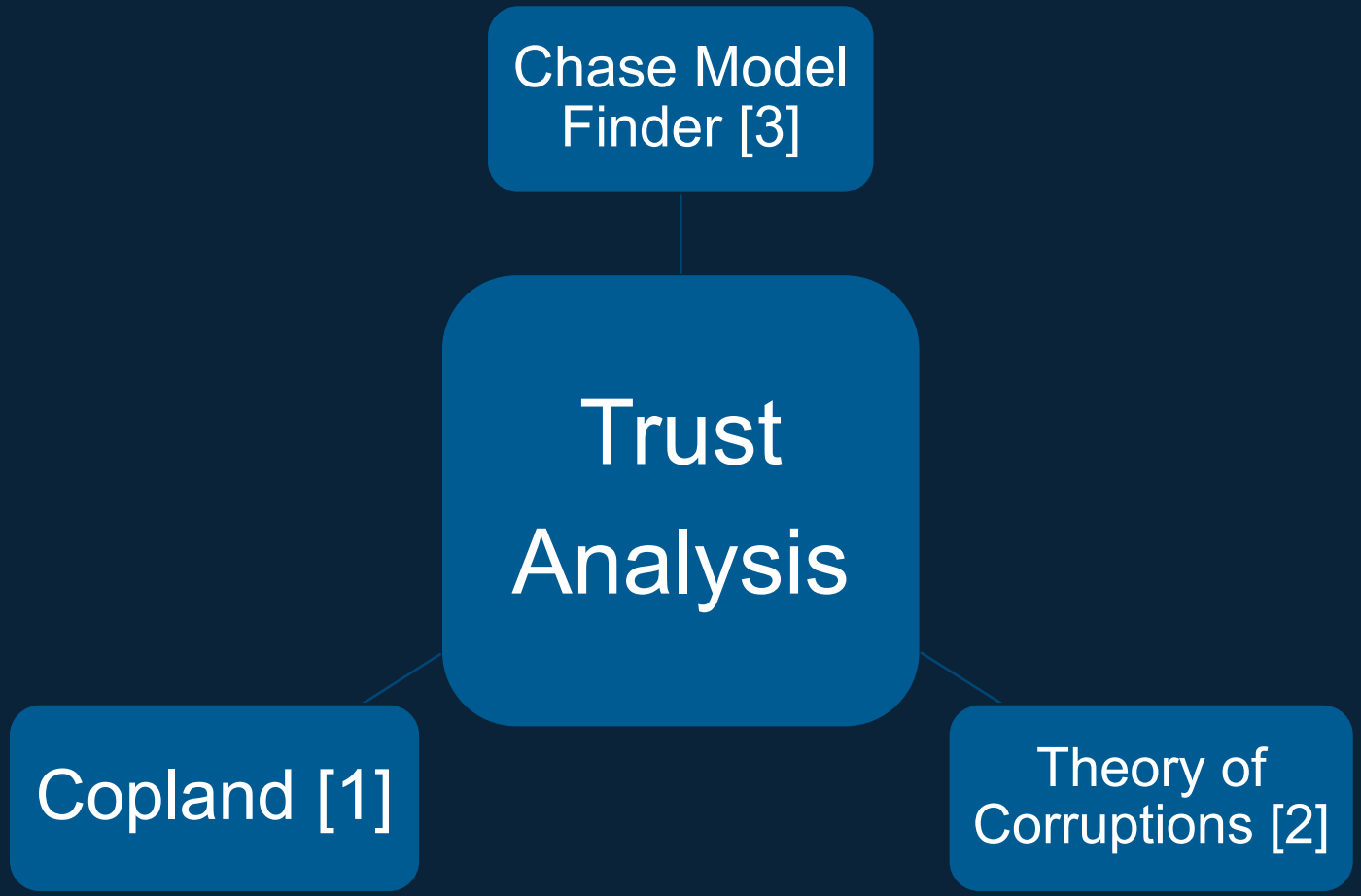
Layered Attestation Example

- Bank wants proof client has no malicious extensions
- When should we trust the browser monitor?
  - When better protected antivirus says so
- But when should we trust the antivirus?
  - When even better protected kernel integrity monitor says so
- With more layers, undetected corruptions harder for adversary



Layered attestations use layered dependencies to increase trust in integrity evidence

Relation to Prior Work



[1] John D. Ramsdell, Paul D. Rowe, Perry Alexander, Sarah C. Helble, Peter Loscocco, J. Aaron Pendergrass, and Adam Petz. Orchestrating layered attestations, In Principles of Security and Trust, 2019. Springer.
[2] Paul D. Rowe. Confining adversary actions via measurement. In Graphical Models for Security, 2016. Springer.
[3] John D. Ramsdell. Chase Source Repository. The MITRE Corporation, 2019. https://github.com/ramsdell/chase

Objective

We have

Copland, a means of specifying layered attestations,

and

a theory of runtime corruption

and its effects on evidence appraisal.

We want to

automatically reveal undetected corruptions

permitted by a Copland phrase.

Copland Specification Language

- Atomic actions

C1 P C2	C1 measures C2 at place P
HSH	Hash evidence
SIG	Sign evidence with my key
…	Other actions

Language is parametric w.r.t. atomic actions

- Remote request

$@_P$ T	Perform T at P

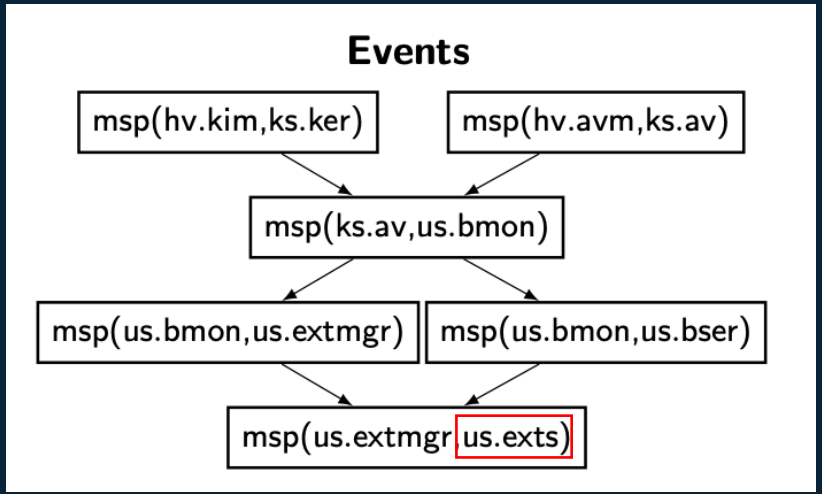
- Combining actions

$\cdot \in \{+, -\}$

$t_1 \cdot \rightarrow \cdot t_2$	Sequential composition with data passing
$t_1 \cdot \prec \cdot t_2$	Sequential composition without data passing
$t_1 \cdot \sim \cdot t_2$	Parallel composition

Copland Event Semantics

*bank : @$_{hv}$[(kim ks ker +~+ avm ks av)
+<+ @$_{ks}$[av us bmon
+<+ @$_{us}$[(bmon us extmgr +~+ bmon us bser)
+<+ extmgr us exts]]]



Challenge: Determining necessary adversary activity required to ensure undetected corruption

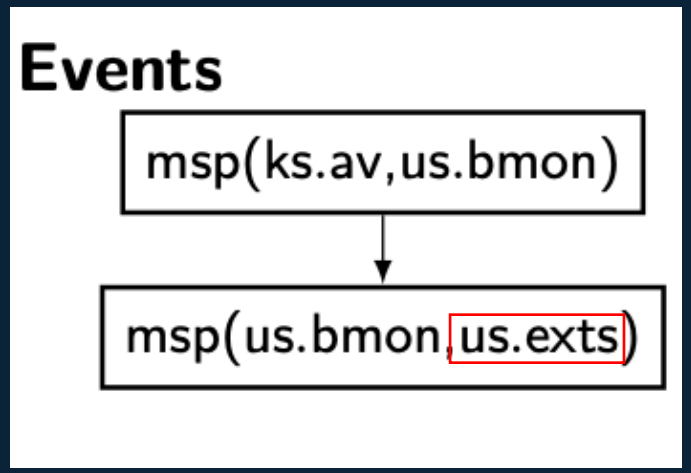
Adversarial Setting

- Copland event semantics enriched with corrupt and repair events
- Corrupt measurers' evidence always passes appraisal
- Regular measurers' evidence is always accurate
- Adversary always has strategies to "win"
  - Harder: Recent and deep corruptions
    - Narrow timeframes, better-protected components
  - Easier: Shallow corruptions, repairs

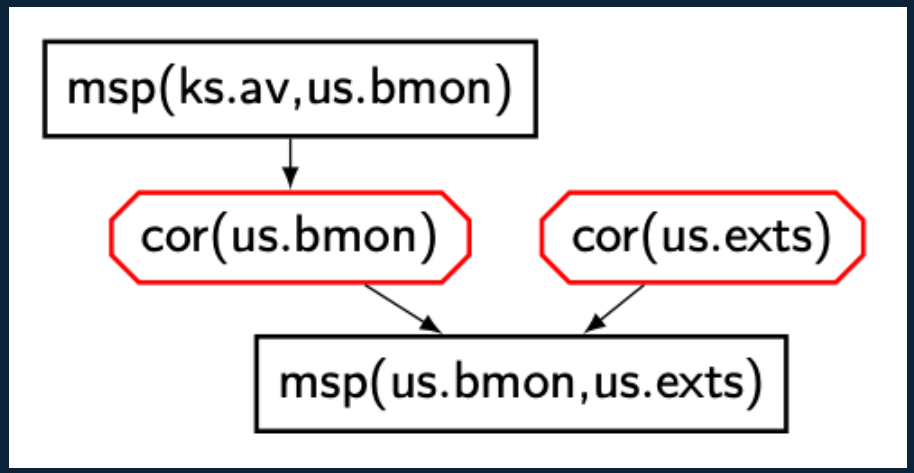
Big question: How easily can the adversary defeat the attestation via undetected corruptions?

Example: Sequential measurements

*bank : $@_{ks}$[av us bmon] + < + $@_{us}$[bmon us exts]



Assumption

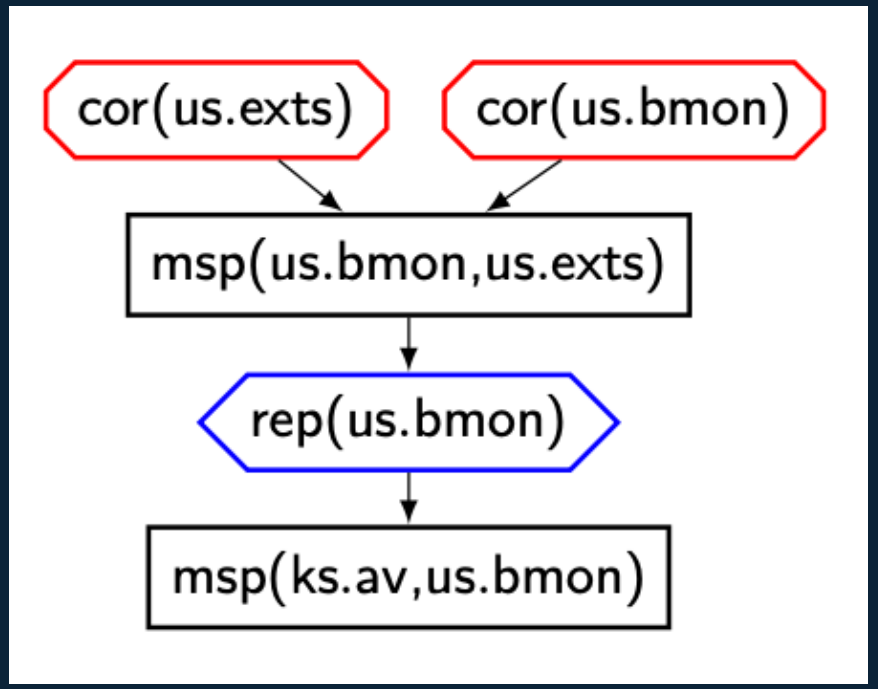


Possible conclusion

Example: Parallel measurements

*bank : $@_{ks}$[av us bmon] + ~ + $@_{us}$[bmon us exts]

Assumption



Possible conclusion

Moral: Some ways of composing measurements make undetected corruptions harder to achieve

Recall: Objective

We want to

automatically enumerate all models with

undetected corruptions permitted by a Copland phrase.

Strategy

Axiomatize the theory of corruption and repair and

submit theory to model finder to enumerate adversarial executions

Chase Model Finder

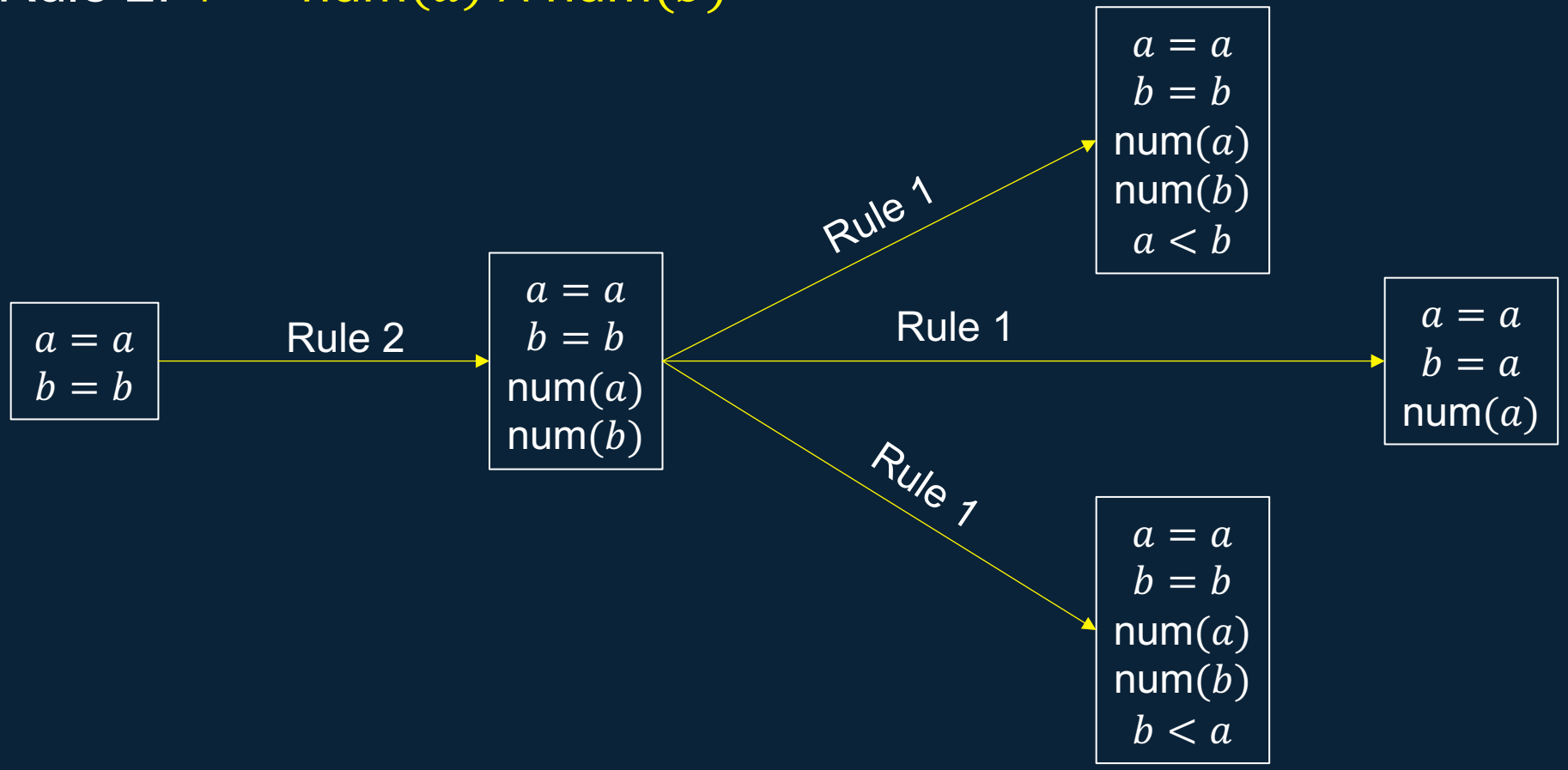
- Model finder for geometric logic
  - The subset of first-order logic preserved under homomorphism
  - Assumes formulas are in special geometric form

- Finds a set of support
  - A (usually small) characterization of all models of a theory

- Based on an algorithm from database theory
  - Progressively adds structure, branching on disjunctions

Example: Totally ordered sets of at most 2 numbers

- Rule 1: $\forall X, Y\ .\ \mathrm{num}(X) \wedge \mathrm{num}(Y) \Rightarrow X = Y \vee X < Y \vee Y < X$
- Rule 2: $\top \Rightarrow \mathrm{num}(a) \wedge \mathrm{num}(b)$



Saturation: A correctness criterion

- Models are pairs (E, φ)
  - E a partially-ordered set of events
  - φ a set of assumptions about the corrupt components at events in E
- (E, φ) is saturated if
  - It detects no corruptions
  - E and φ are "mutually consistent"

Theorem
*The minimal saturated models extending an input query are a **set of support** for executions in the denotation of the query.*

- Good news: Looking for set of support. We can use Chase!
- Bad news: Axiomatization is not in special geometric form needed by Chase.

Converting to Special Geometric Form

Theorem
The model (E, φ) is saturated iff Formulas 1—4 are all satisfied.

`% Formula 1`

$\forall \boldsymbol{e}, \boldsymbol{p}, \boldsymbol{m}, \boldsymbol{q}, \boldsymbol{t} \,.\, \ell(\boldsymbol{e}) = \text{msp}(\boldsymbol{p}.\boldsymbol{m}, \boldsymbol{q}, \boldsymbol{t}) \wedge \boldsymbol{\varphi}(\boldsymbol{q}.\boldsymbol{t}, \boldsymbol{e})$

$\Rightarrow \boldsymbol{\varphi}(\boldsymbol{p}.\boldsymbol{m}, \boldsymbol{t}) \vee \exists \boldsymbol{c} \,.\, \text{Depends}(\boldsymbol{p}.\boldsymbol{m}, \boldsymbol{p}.\boldsymbol{c}) \wedge \boldsymbol{\varphi}(\boldsymbol{p}.\boldsymbol{c}, \boldsymbol{e})$

`% Formula 2`

$\forall \boldsymbol{e}, \boldsymbol{p}, \boldsymbol{c} \,.\, \boldsymbol{\varphi}(\boldsymbol{p}.\boldsymbol{c}, \boldsymbol{e})$

$\Rightarrow \exists \boldsymbol{e}' . \boldsymbol{e}' \prec \boldsymbol{e} \wedge \ell(\boldsymbol{e}') = \text{cor}(\boldsymbol{p}.\boldsymbol{c})$

`% Formula 3`

$\forall \boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{p}, \boldsymbol{c} \,.\, \boldsymbol{e_1} \prec \boldsymbol{e_2} \wedge \boldsymbol{\varphi}(\boldsymbol{p}.\boldsymbol{c}, \boldsymbol{e_2}) \wedge \ell(\boldsymbol{e_1}) = \text{rep}(\boldsymbol{p}.\boldsymbol{c})$

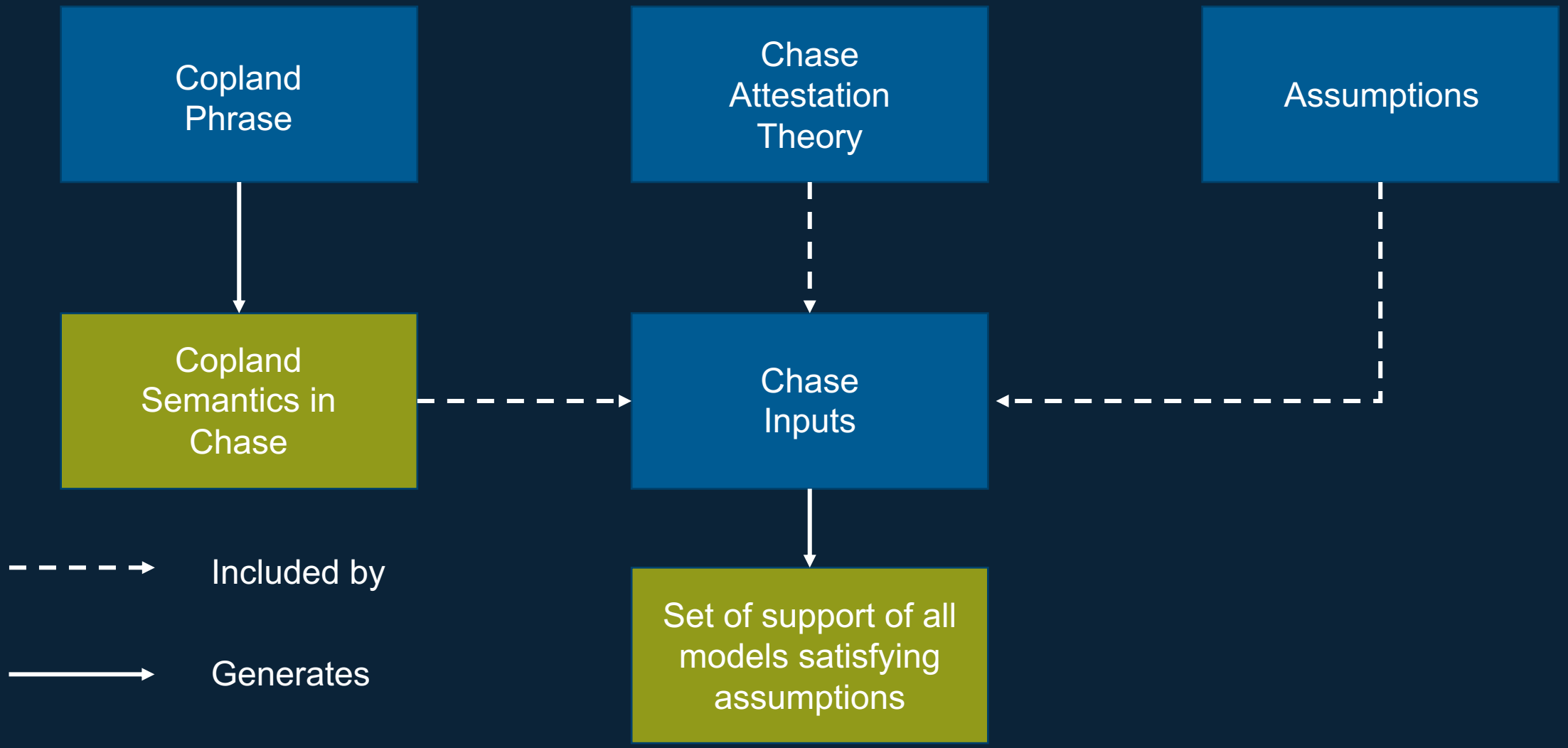
$\Rightarrow \exists \boldsymbol{e}' . \boldsymbol{e_1} \prec \boldsymbol{e}' \wedge \boldsymbol{e}' \prec \boldsymbol{e_2} \wedge \ell(\boldsymbol{e}') = \text{cor}(\boldsymbol{p}.\boldsymbol{c})$

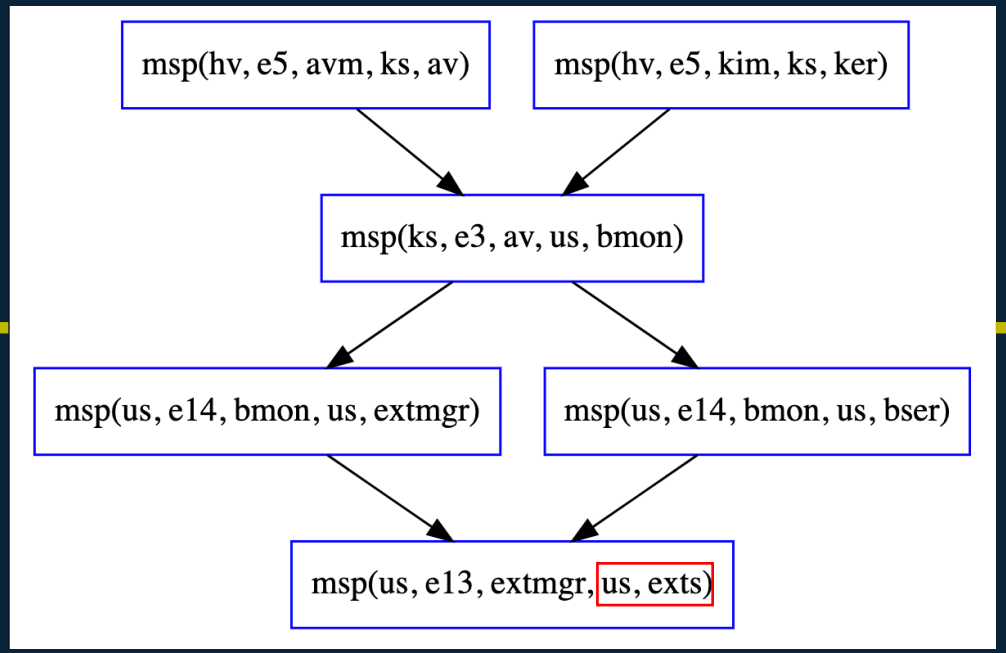
`% Formula 4`

$\forall \boldsymbol{e_1}, \boldsymbol{e_2}, \boldsymbol{p}, \boldsymbol{c} \,.\, \boldsymbol{e_1} \prec \boldsymbol{e_2} \wedge \ell(\boldsymbol{e_1}) = \text{cor}(\boldsymbol{p}.\boldsymbol{c}) \wedge \text{ms\_event}(\boldsymbol{e_2}) \wedge \text{relevant}(\boldsymbol{p}.\boldsymbol{c}\, \boldsymbol{e_2})$

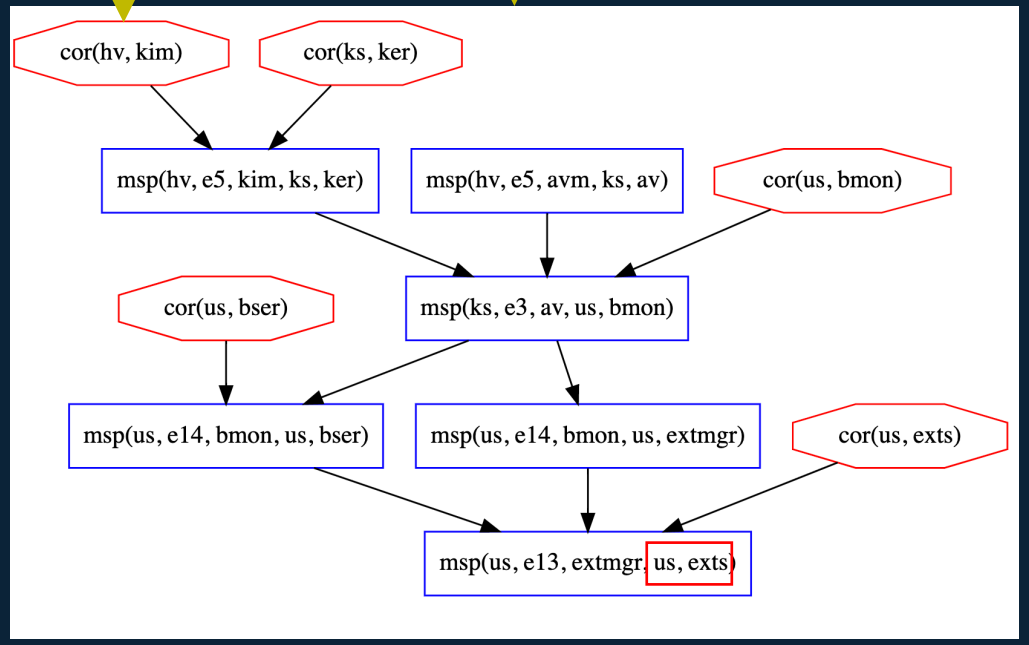
$\Rightarrow \boldsymbol{\varphi}(\boldsymbol{p}.\boldsymbol{c}, \boldsymbol{e_2}) \vee \exists \boldsymbol{e}' . \boldsymbol{e_1} \prec \boldsymbol{e}' \wedge \boldsymbol{e}' \prec \boldsymbol{e_2} \wedge \ell(\boldsymbol{e}') = \text{rep}(\boldsymbol{p}.\boldsymbol{c})$

Trust Analysis Workflow





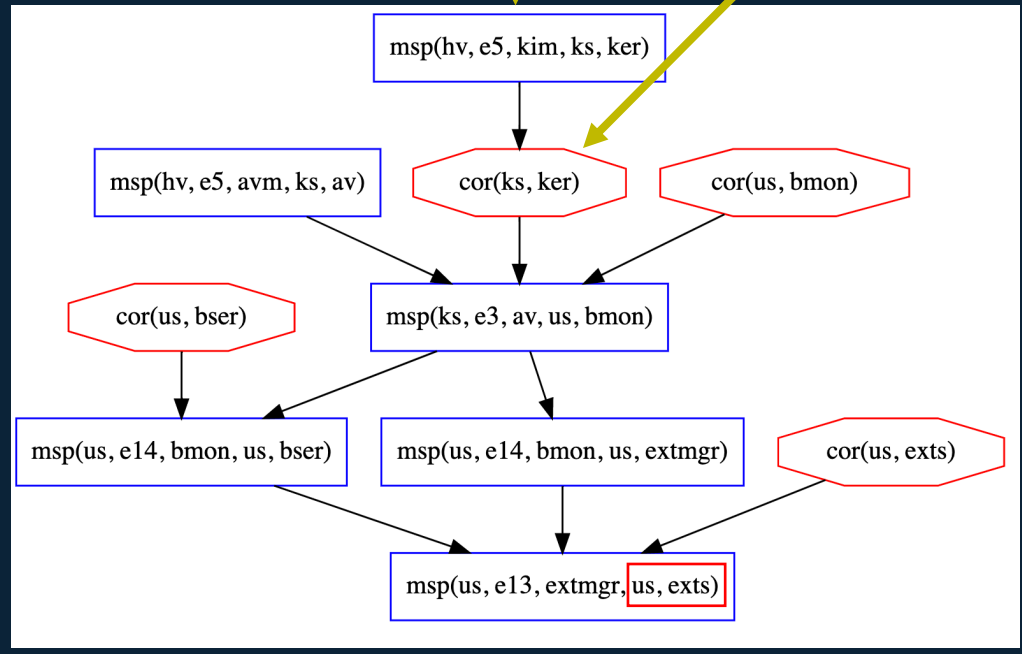
Deep Corruption



Chase finds 40 models!

...

Recent Corruption



Assumptions Represent Acceptable Risks

- At which measurement did adversary avoid detection?
- Don’t show executions in which:
  - Component X is corrupted (Deep)
  - Component Y is corrupted *after* the attestation starts (Recent)
  - Component Z depends on other components for measurements (Context)

- Assumptions are crucial for efficient & terminating searches

Results and Performance

Performance Depends On:

- Details of Copland phrase
  - More parallelism, more models
- Assumptions made
  - Dependency relation
  - Types of behavior excluded
- Chase optimizations
  - Scheduling & minimality

Phrase	Expl. Dep.	Exclude Deep	Exclude Recent	# Models Found	Time
Ex. 1				5	0.53s
Ex. 1	✓			3	0.42s
Ex. 1	✓	✓	✓	1	0.33s
Ex. 2				4	0.48s
Ex. 2	✓			2	0.41s
Ex. 2	✓	✓	✓	0	0.31s
Ex. 17				2,478	4m15.08s
Ex. 17	✓			40	2.64s
Ex. 17	✓	✓	✓	0	0.51s

Files available at: https://copland-lang.org/resources/chase/ppdp/README

Summary

- Layered attestations make undetected corruption harder for an adversary
- Our automated method enumerates all executions with undetected corruptions
- These tools will help trust policymakers evaluate complex alternatives
- Future work will consider adversarial manipulation of data flow

Thank You!!

- Email: `{prowe, ramsdell, ikretz}@mitre.org`
- Copland: `https://copland-lang.org`
  - Contains links to download
    - Analysis pipeline tools
    - Inputs for examples from paper
    - Related resources/projects
- Chase: `https://github.com/ramsdell/chase`
  - Install with: `opam install chase`