Oral Examination: Logics of Specifications And Specification Mining

Calvin Deutschbein

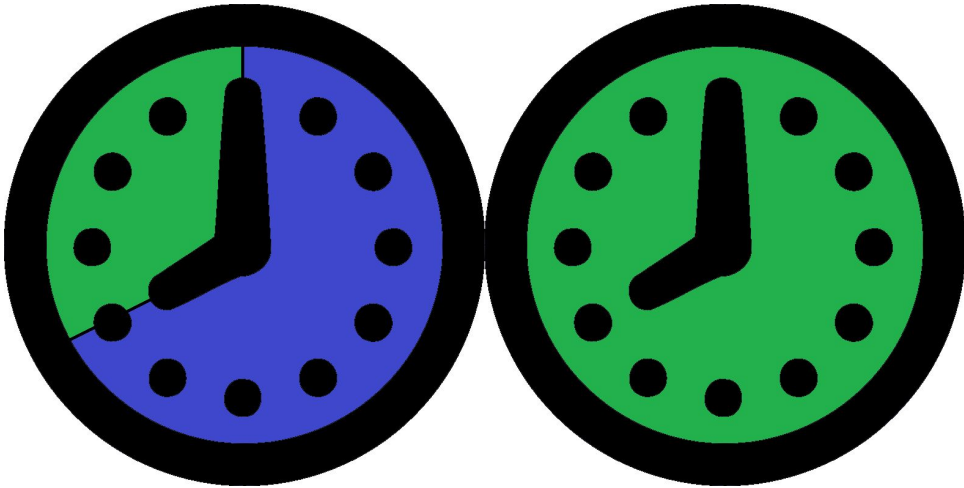
Format

2 hour examination

40-50 minute presentation

10 papers:

- 4 on logics of specifications
- 6 on specification mining



Logics of Specifications

Logics of Specifications

Denning (1976) - Information Flow

Alpern and Schneider (1987) - Safety and Liveness

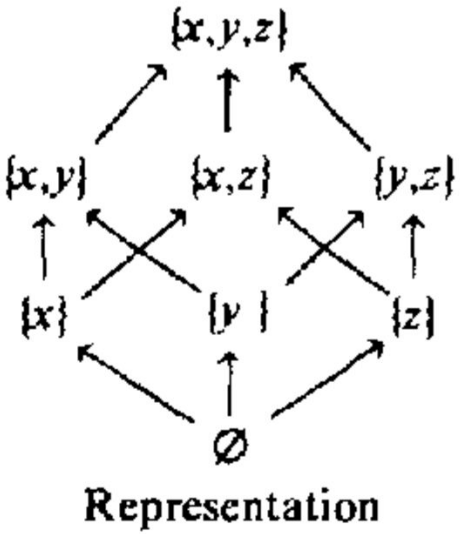
Clarkson and Schneider (2008) - Hyperproperties

Clarkson et al. (2014) - HyperLTL and HyperCTL*

Will borrow from mining papers.

Logics of Specifications

Specifications are properties or hyperproperties that define the behavior of a design or system.



Representation

Various logics of specifications have different levels of expressiveness and usability.

Framing Example: Spectre and Meltdown

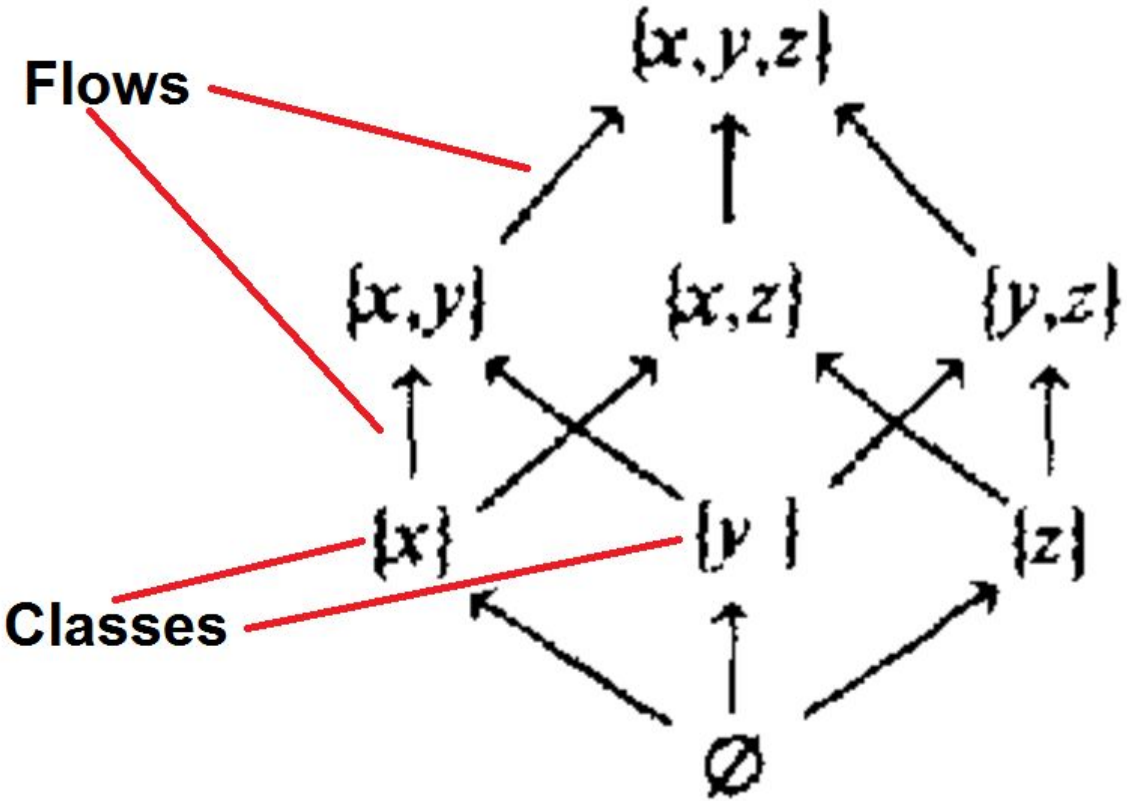
- Information flow
- Temporal relationships
- Multiple traces

Information Flow

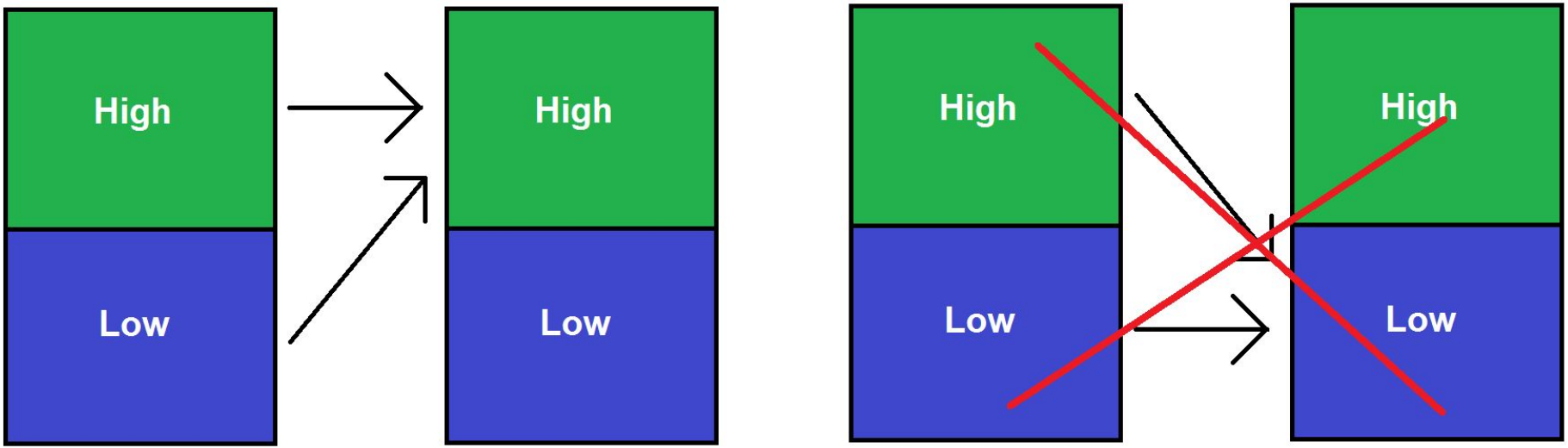
Logics of Specifications

Denning (1976)

Information Flow - Flow Models



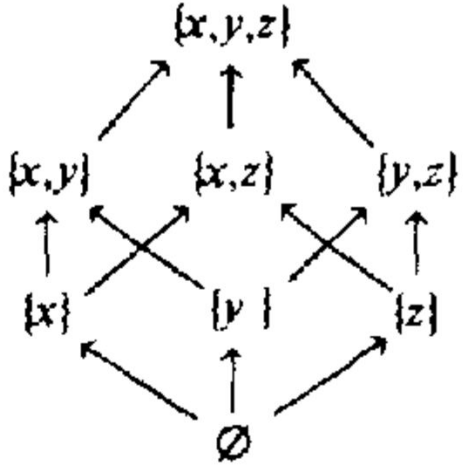
Information Flow Example - High and Low users



Information Flow - Flow Models

FM = (*N*, *P*, *SC*, ⊕, ->).

- *SC* = {*A*, *B*,...}, security classes
  - *N* = {*a*, *b*,...} storage objects
  - *P* = {*p*, *q*,...} processes
- ⊕, class combining operator
- ->, flow relation
  - reflexive, transitive, antisymmetric



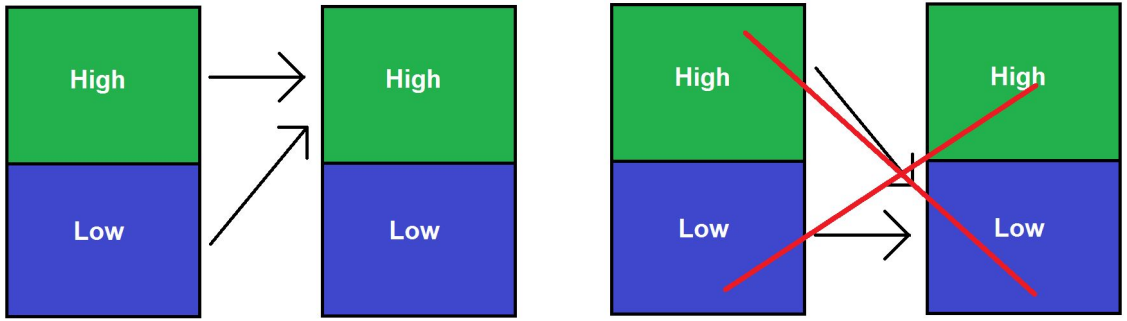
Representation

Information Flow Example - High and Low users

SC = {*H*, *L*} is a set of security classes

- *L* ⊕ *L* = *L* and *H* ⊕ _ = *H*
- *L* -> *H*

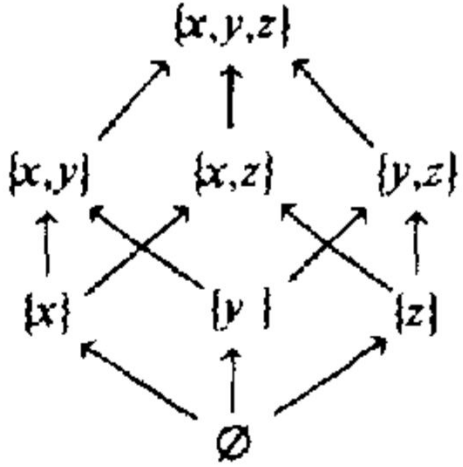
That is, any function with a *H* classified input must have its output classified as *H.*



We can define a lattice

A *universally bounded lattice* is a structure consisting of

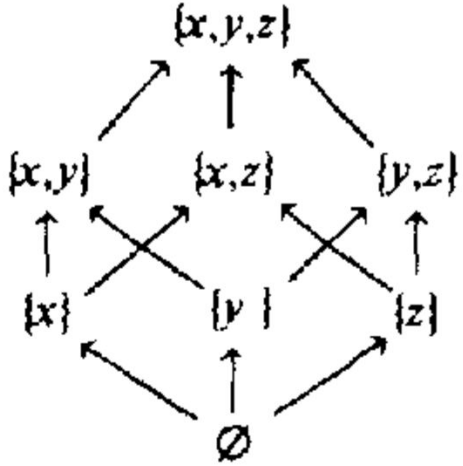
- a finite set
- a partial order
- least upper bounds
- greatest lower bounds



Representation

We can define a lattice

1. *SC* is finite
2. (*SC*, ->) partially ordered set
3. ⊕ is a least upper bound operator on *SC*
4. *SC* has a lower bound *L*



Representation

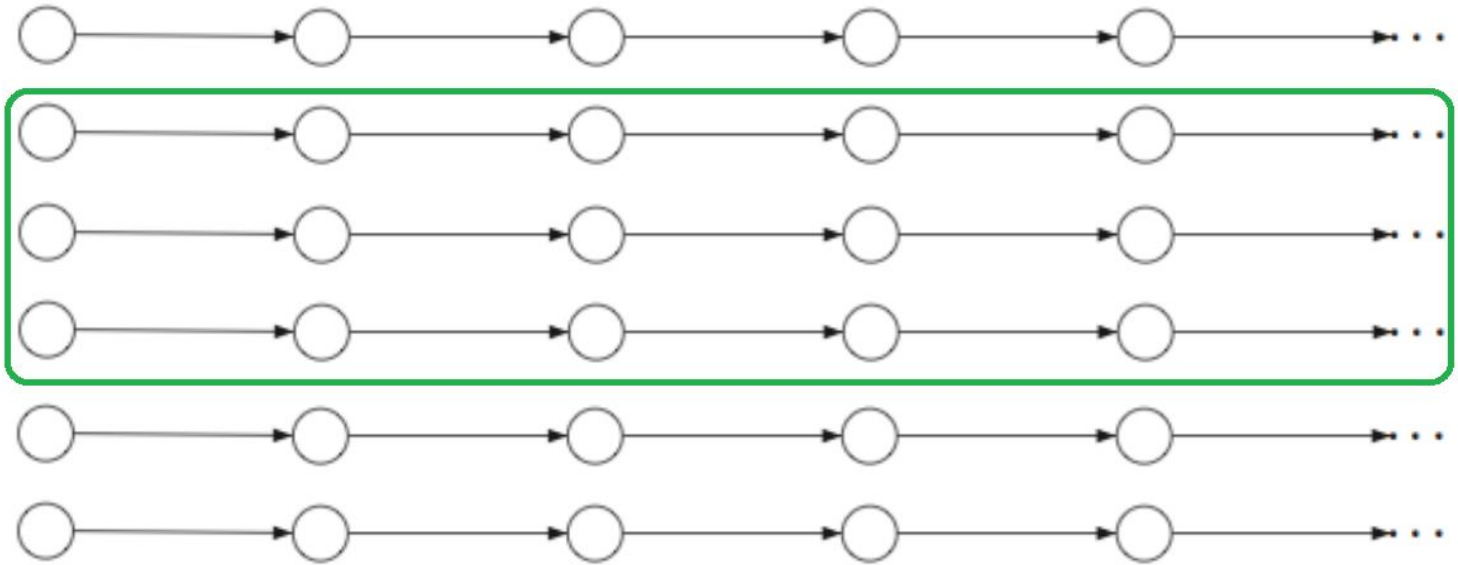
Temporal Logics

Logics of Specifications

Clarkson et al. (2014), Lemieux et al. (2015)

Temporal Logics here define Trace Properties

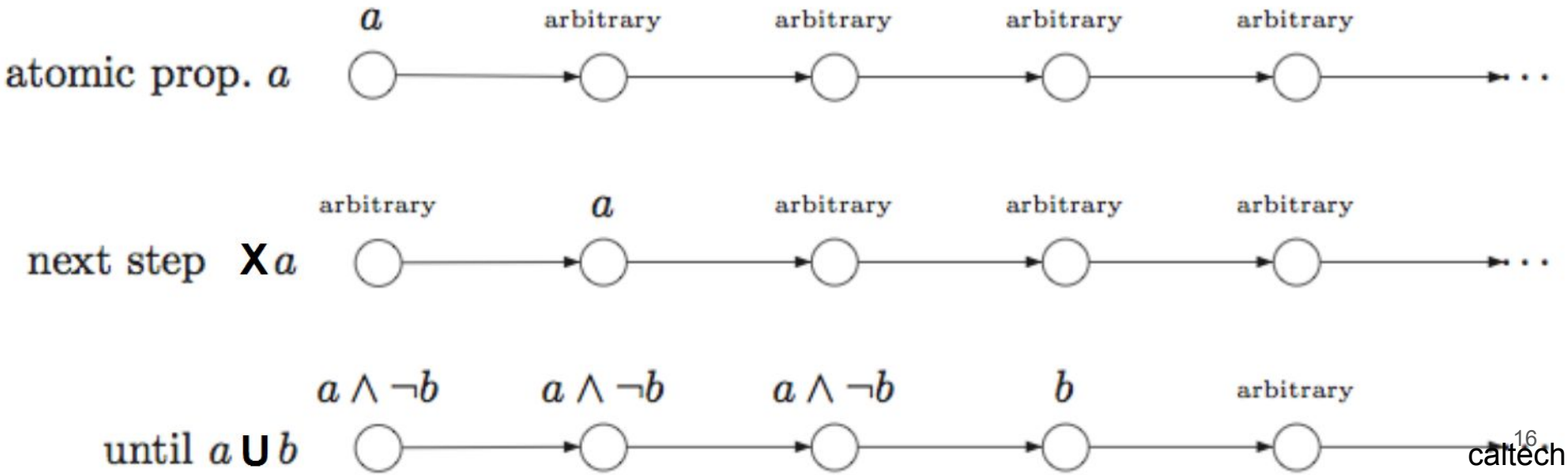
A **trace property** is a set of infinite sequences of program states (a set of traces).



Linear Temporal Logic (LTL) and Trace Properties

Boolean operators “not” and “or”

Temporal operators “next” (**X**) and “until” (**U**).



caltech

Linear Temporal Logic (LTL)

X a — “a” holds in the next time

F a — “a” holds in some future time (true **U** a)

G a — “a” holds in all future times ¬(true **U** ¬a)

a **U** b — “a” holds unless “b” and “b” must hold in the future

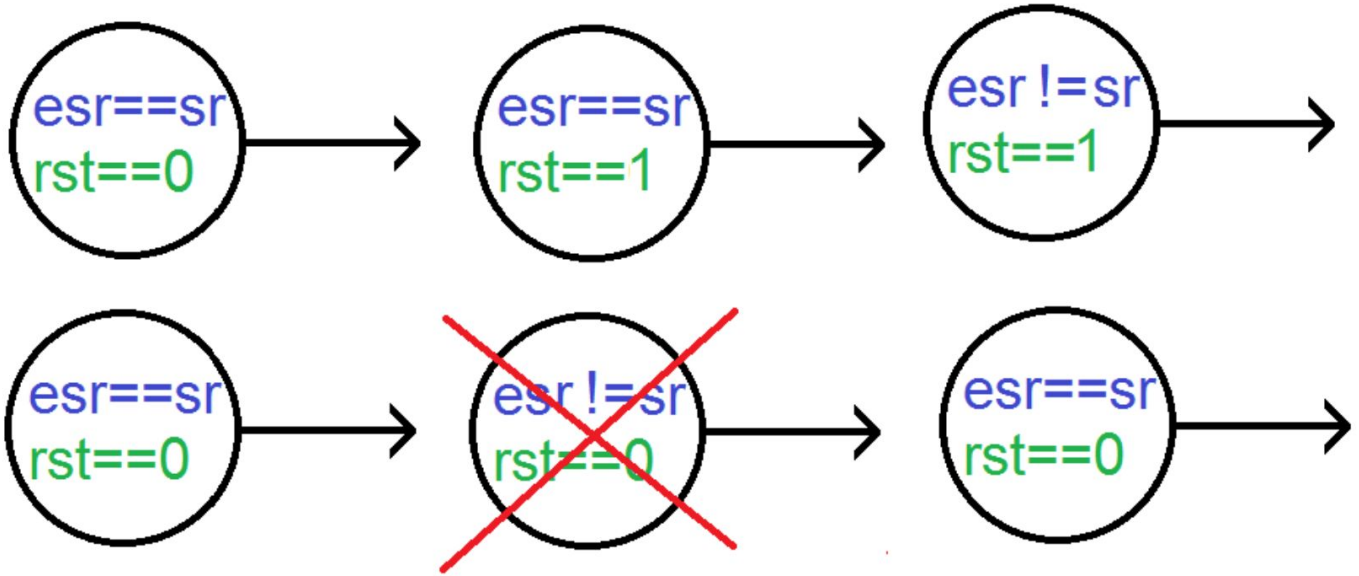
a **R** b — “b” holds until “a” holds forever (b **U** a AND b) OR **G** a

a **W** b — “a” holds unless “b” or “a” holds forever (a **U** b) OR **G** a

a **M** b — “a” holds unless “b” or “a” holds forever b **U** (a AND b)

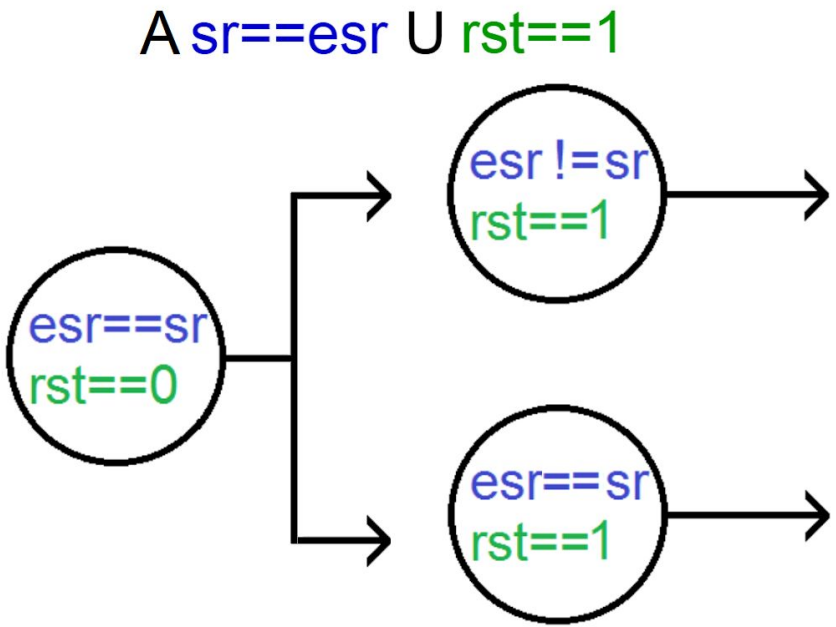
Linear Temporal Logic (LTL)

sr==esr U G(rst==1)



Computational Tree Logic

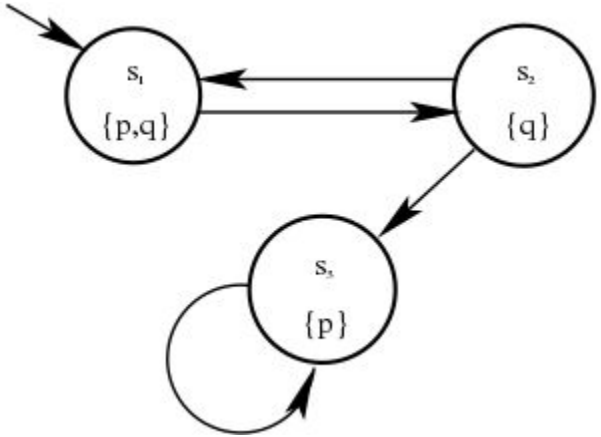
Prefix “all” (**A**) or “exists” (**E**) onto “next” (**X**) and “until” (**U**).



CTL* is the superset of LTL and CTL

CTL* can combine temporal operators after prefix (**AXG** b)

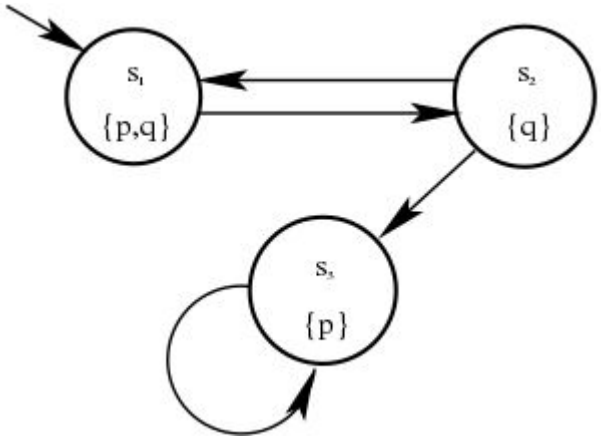
LTL lifts to CTL* by prefixing with **A**



CTL*

CTL* can be captured in Kripke structure:

- *S*, finite set of states
- *I*, initial state subset of *S*
- R, transition relation
- atomic proposition state labels



Safety and Liveness

Logic of Specifications

Alpern and Schneider (1987)

Trace Properties and Büchi Automata

Properties can be expressed as Büchi automata.

Properties can be *safety* or *liveness.*

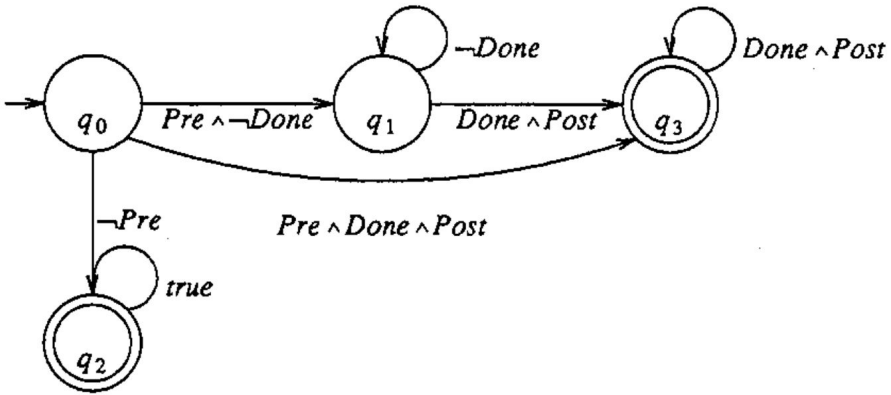


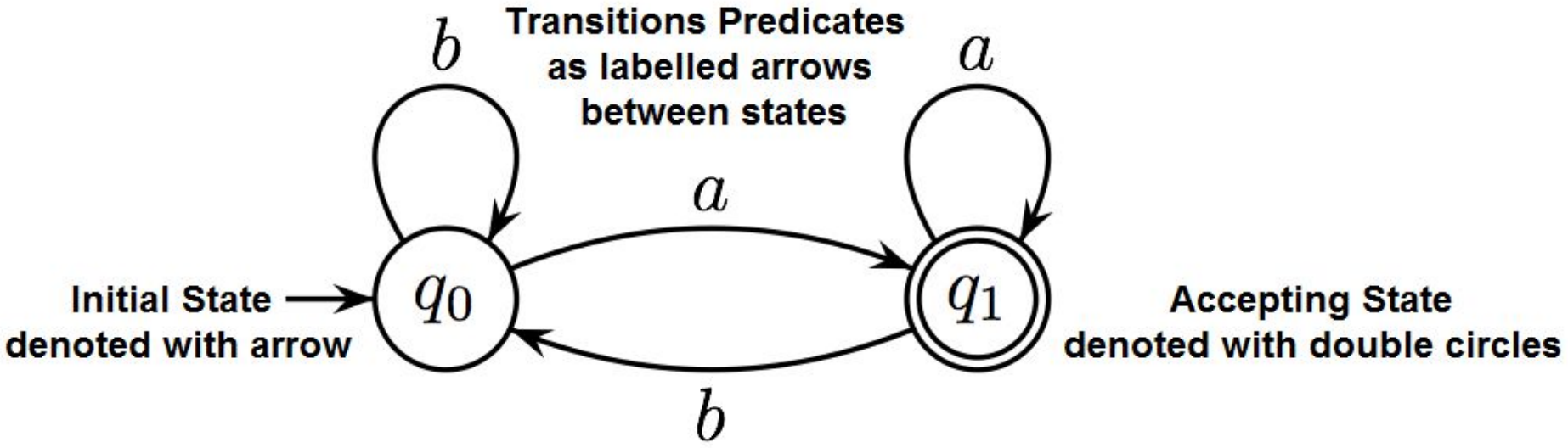
Fig. 1. m_{tc}

Temporal Logics and Büchi Automata

“In fact, Buchi [sic] automata are more expressive than most temporal logic specification languages - there exist properties that can be specified using Buchi automata but cannot be specified in (standard) temporal logics.”

-Alpern and Schneider

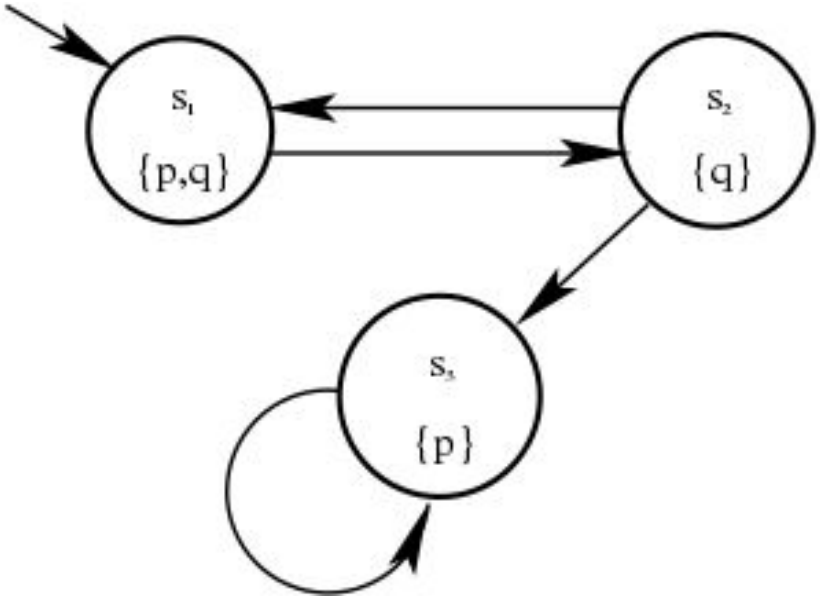
Büchi Automata

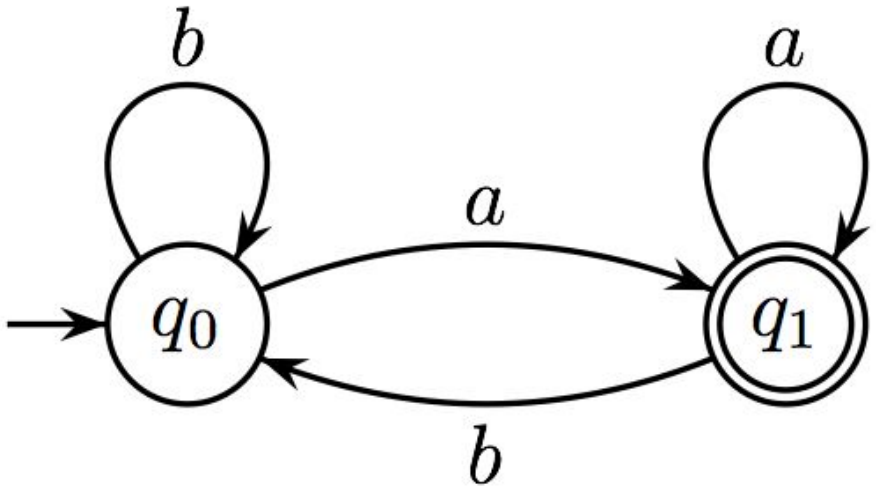


A note: Büchi Automata v Kripke Structures

Büchi automata have accepting states.

Büchi automata have edge labels not state labels.

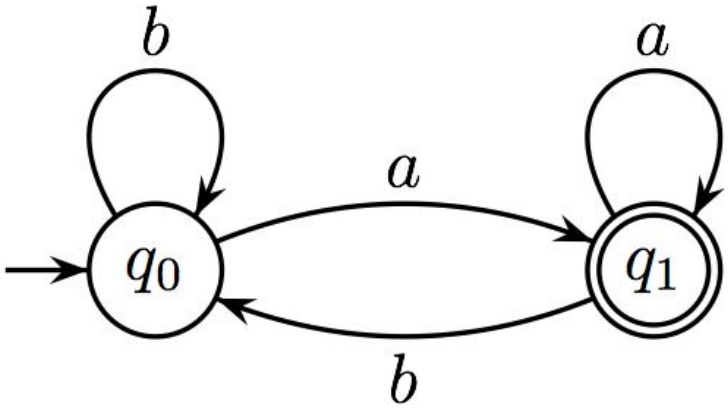




Büchi Automata

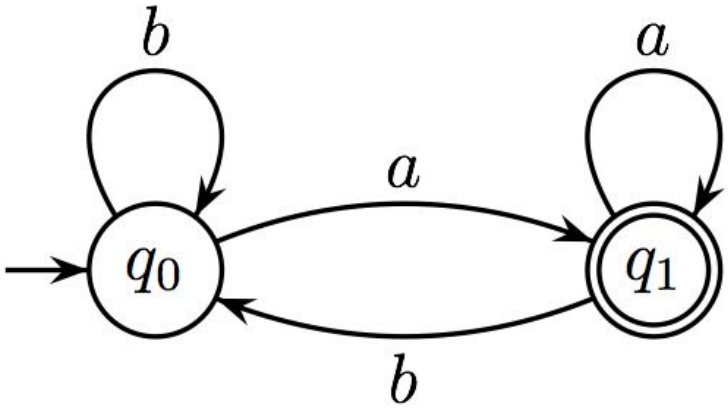
Automata have:

- States
  - Initial state or states
  - Accepting state or states
- Transition predicates
  - May be deterministic or non-deterministic

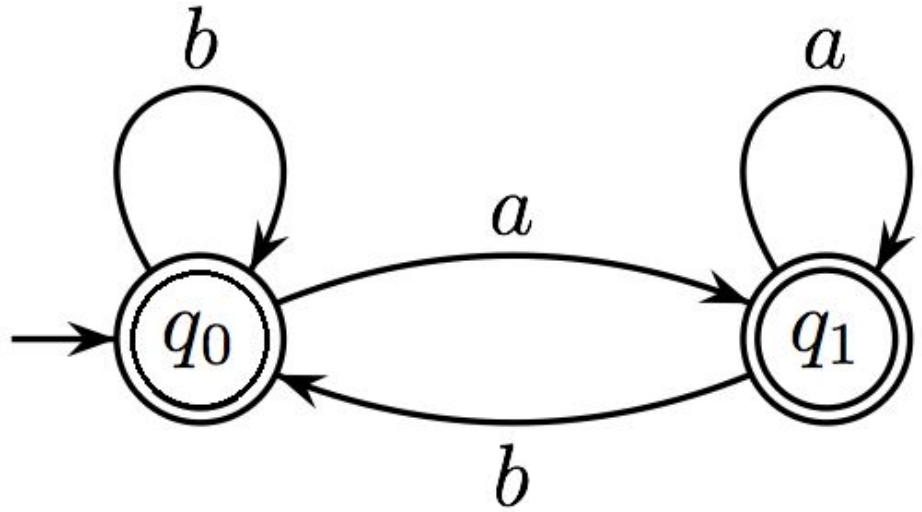


Büchi Automata

A trace is *accepted* if there are no undefined transitions and the trace ends in an accepting state.



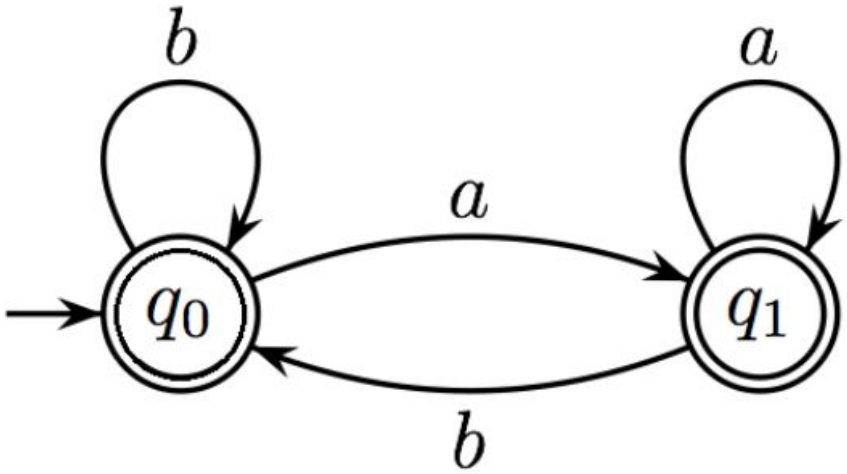
Büchi Automata: *Closure* means all states accept



Safety: Bad Things Do Not Happen

Over a finite trace prefix (that contains bad thing)

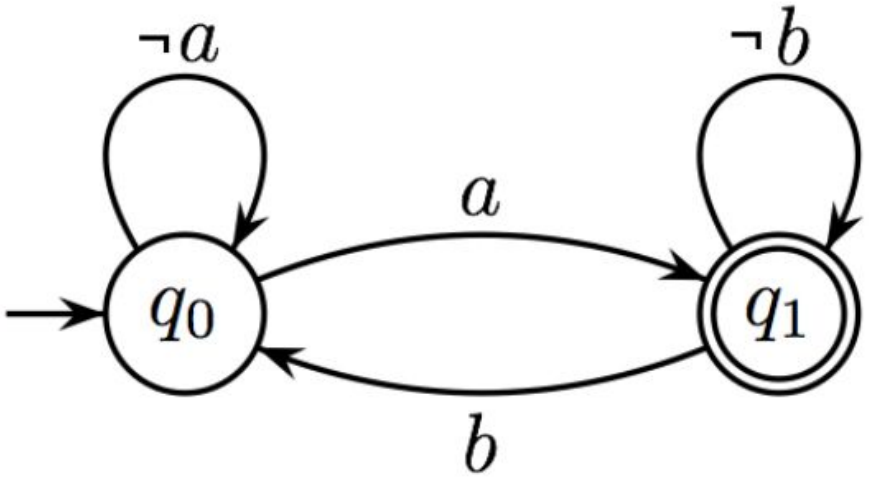
If an automata is its own closure, it defines a safety property!



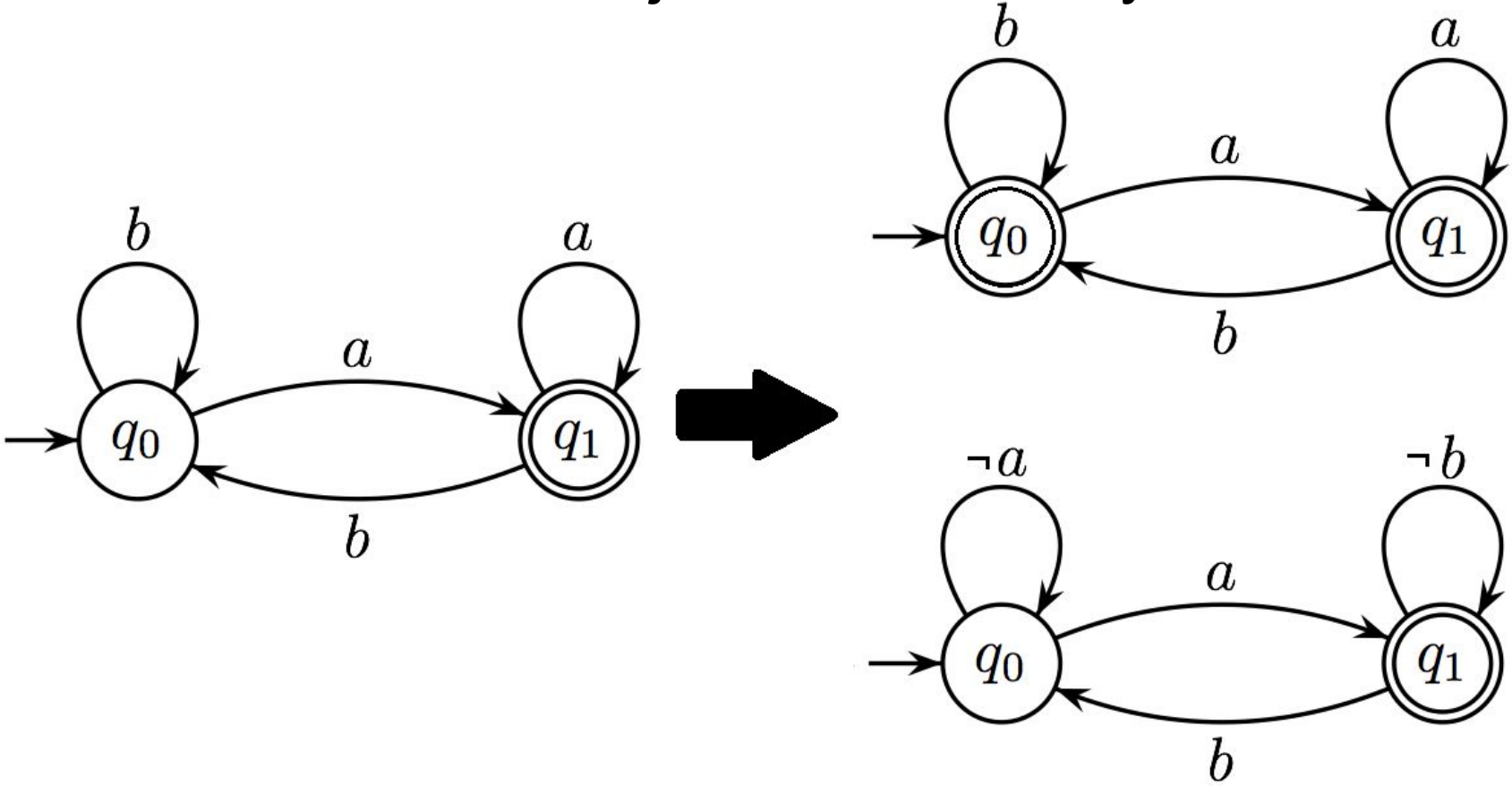
Liveness: Good Things Do Happen

May be over infinite traces (good things keeps happening)

If a closure accepts all traces, it defines a liveness property!



An automata is a conjunction of safety and liveness



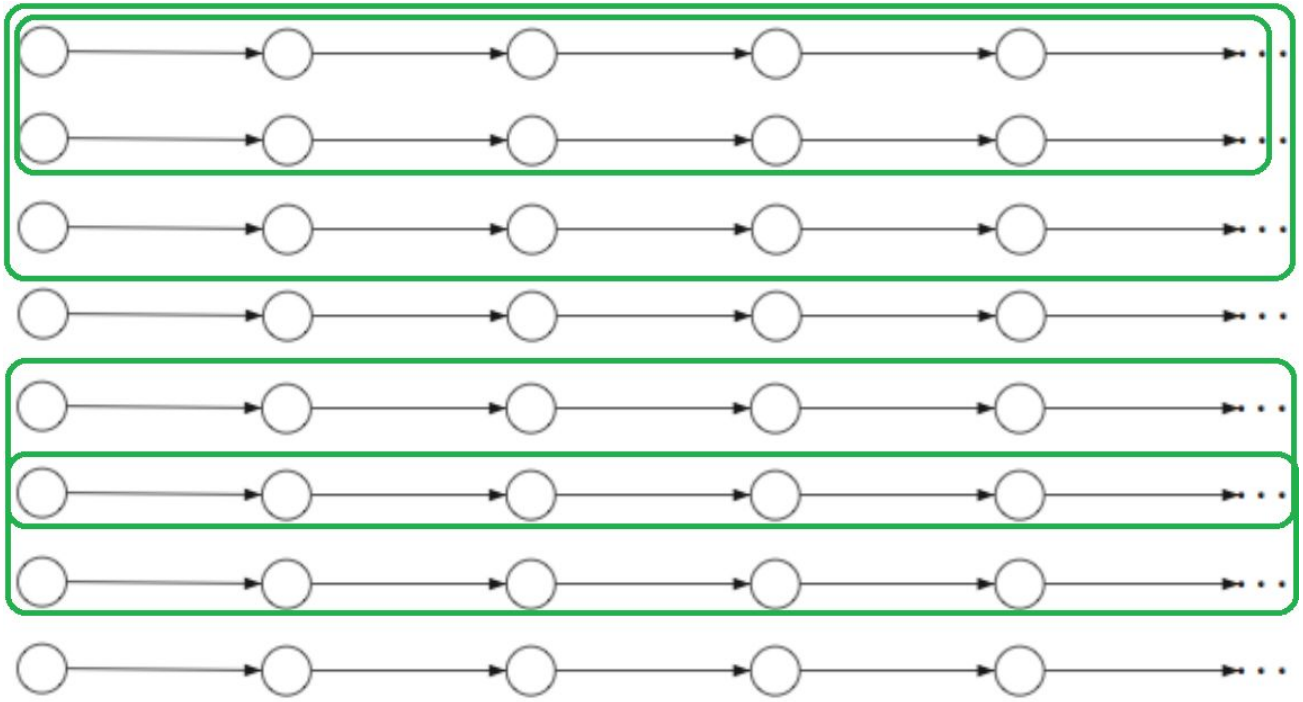
Hyperproperties

Logic of Specifications

Clarkson and Schneider (2008) and Smith et al. (2017)

Hyperproperties

Sets of Sets of Traces, or Sets of Properties



Expressiveness

“We have not been able to find requirements on system behavior that cannot be specified as a hyperproperty.”

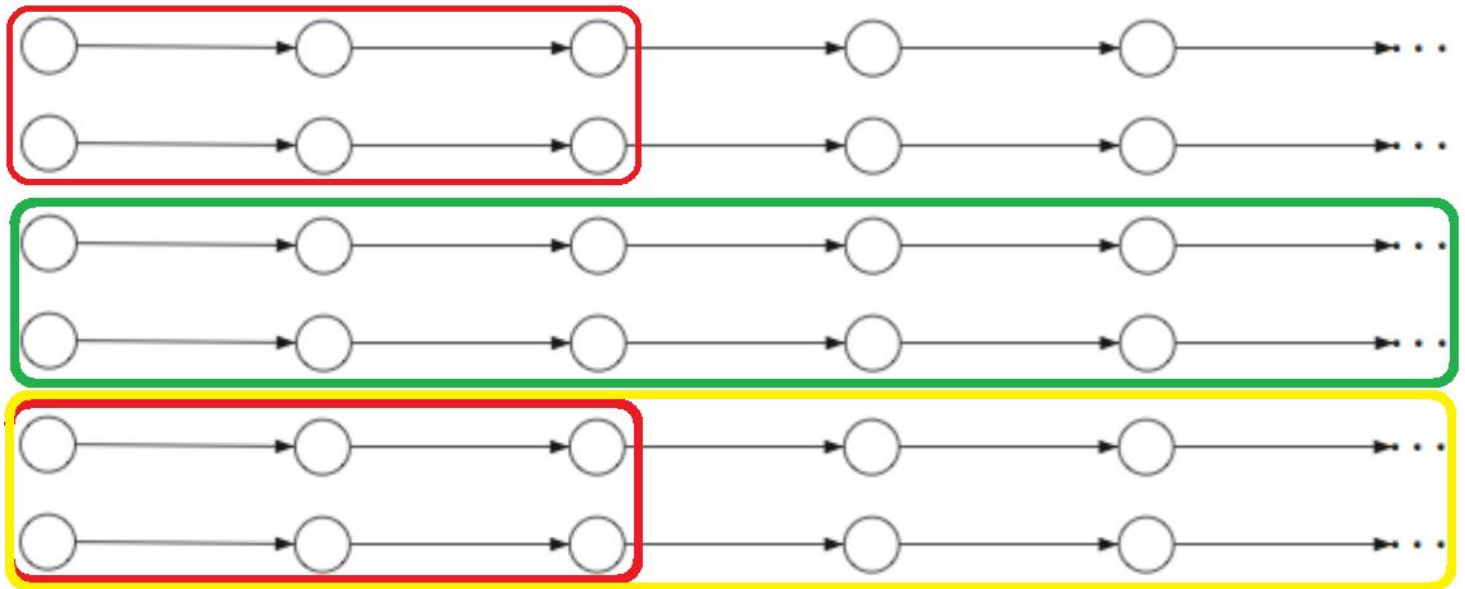
“It is natural to ask whether introducing yet one more level of sets might also be useful. We believe it is not.”

-Clarkson and Schneider

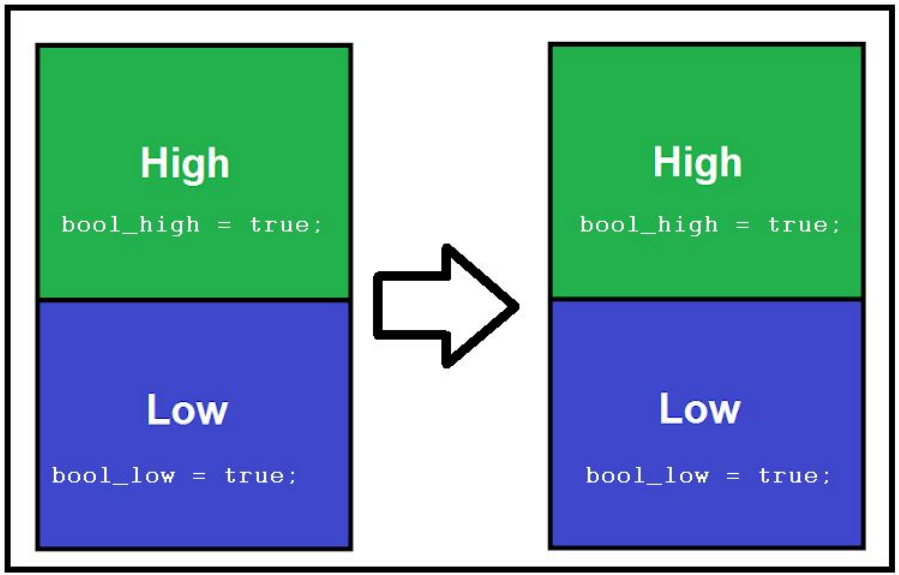
Hypersafety

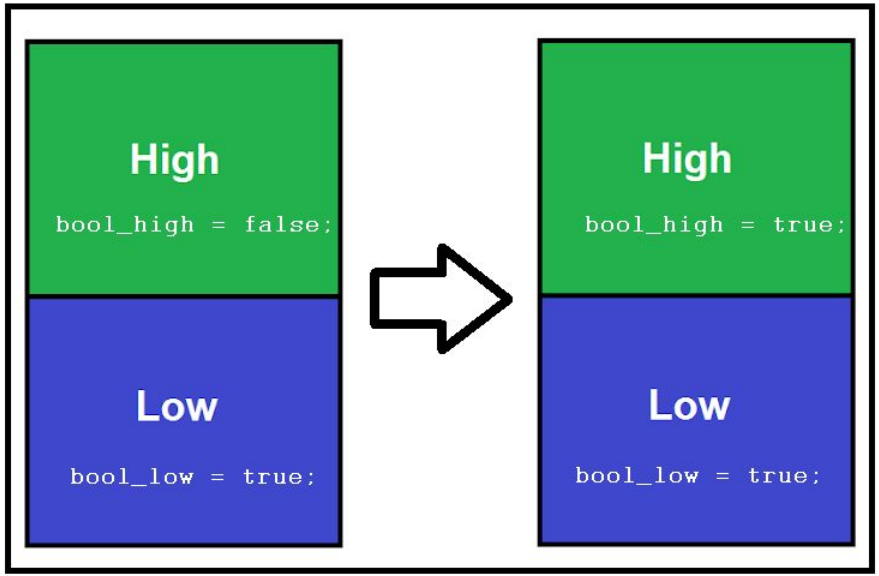
Define $T \leq T'$ to mean $\forall t \in T : (\exists t' \in T' : t \leq t')$

T not in $\boldsymbol{S}$ means $\exists\ M \leq T$, s.t. $\forall T' : M \leq T' \Rightarrow T'$ not in $\boldsymbol{S}$



Example: GMNI (Noninterference)





Example: GMNI (Noninterference)

Safety:

We a concerned about a bad thing happening.

Hyperproperty:

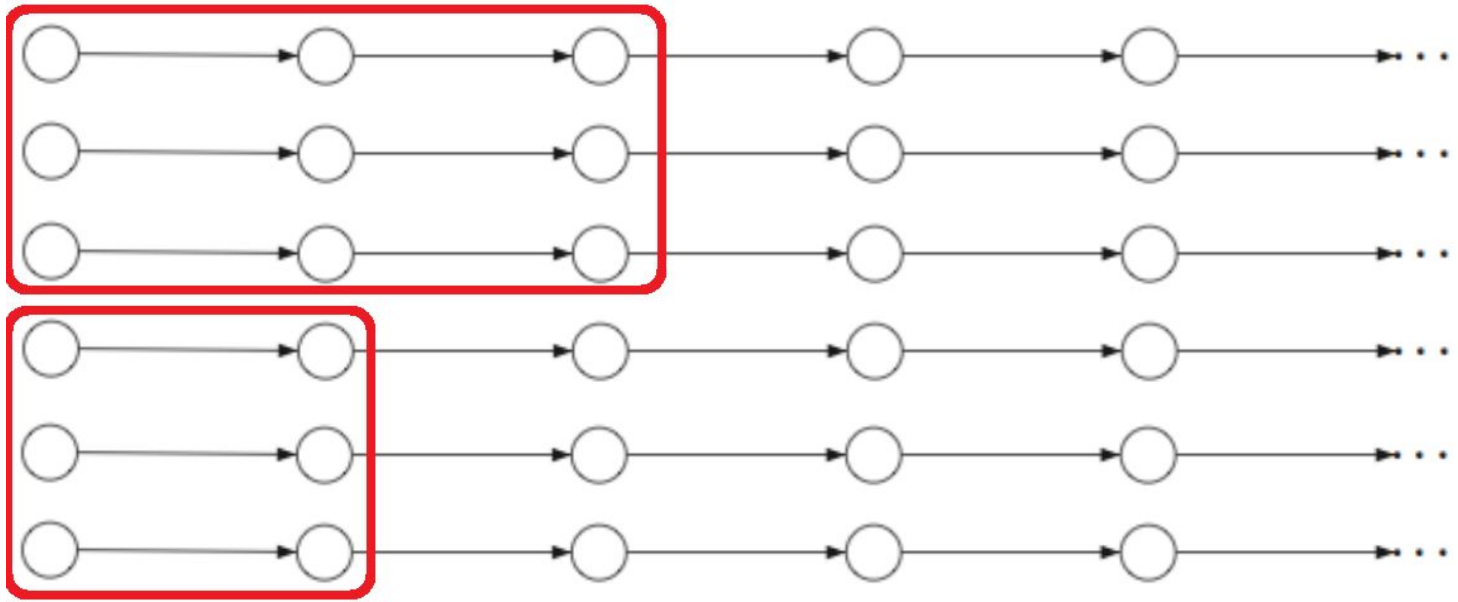
We need two traces.

GMNI is given by T s.t.

$$\forall\ t \in T : \exists\ t' \in T : t =_{\mathsf{L}} t'$$

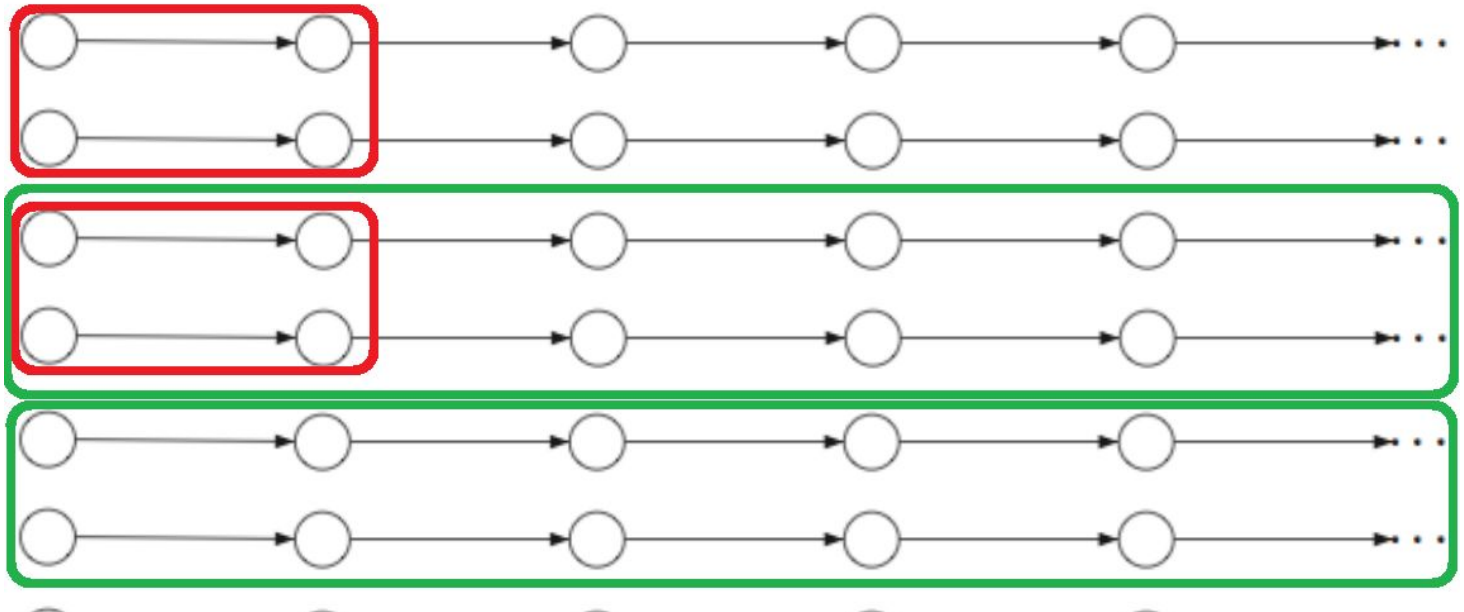
k-Safety, often 2-safety

$$\exists\ M \leq T\ \wedge\ |M| \leq k, \text{ s.t. } \forall\, T'\ : M \leq T' \Rightarrow\ T' \text{ not in } \boldsymbol{S}$$



Hyperliveness

$$(\forall\, T : (\exists\, T' : T \leq T' \wedge T' \in \boldsymbol{L}))$$



Example: RT (Average Response Time)

Liveness:

We are concerned about a good thing happening

Hyperproperty:

We consider all traces within a set.

RT is given by T s.t.

The average response time over all executions in T is less than some value.

Relational Hyperproperties/Specifications

Systems can be modeled as having only beginning and ending states.

These create traces of length two.

Consider RNI, relational non-determinism.

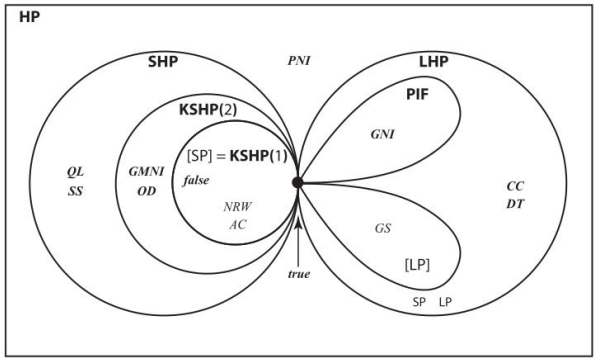
$$\forall\ t, t' \in T : t\,[0] =_L t'\,[0] \Rightarrow t\,[1] =_L t'\,[1]$$

Hyperproperties

Every hyperproperty is some combination of a hypersafety and hyperliveness.

Only *true* is both hypersafety and hyperliveness.

“Finitely observable” forms a topology.



Temporal Hyperproperties

Logics of Specifications

Clarkson et al. (2014), Lemieux et al. (2015)

HyperLTL

∀ t

∃ t

X a

a **U** b

¬ a

a **∨** b

Path modifiers must be prefixes!

… and atomic propositions!

Basically LTL with the ability to define multiple traces.

HyperLTL

We can formulate hyperproperties in HyperLTL.

Observational Determinism:

$$\forall T. \forall T' \,.\, T[0] =_L T'[0] \rightarrow T =_L T'$$

Declassification:

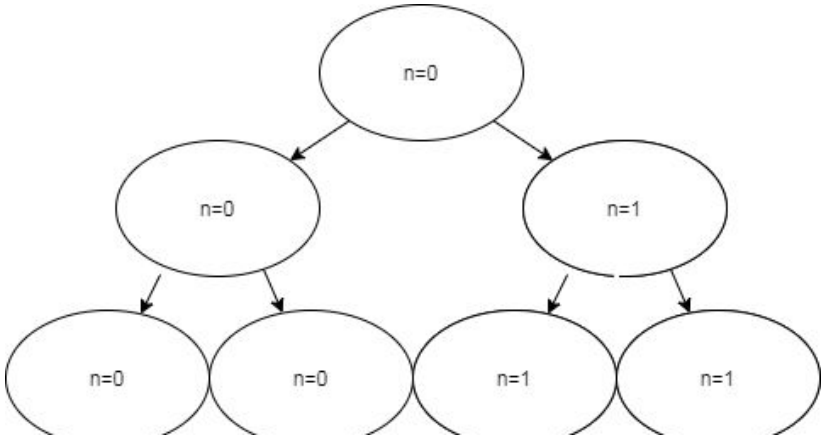
$$\forall T. \forall T' \,.\, (T[0] =_L T'[0] \wedge X(pw_T \leftrightarrow pw_{T'})) \rightarrow T =_L T'$$

HyperCTL*

HyperCTL* generalizes HyperLTL.

$$\forall T.\ \mathbf{X}\ \forall T'.\ \mathbf{X}\ (n = n')$$

“A low user not being able to determine the second branch.”



Framing Example: Spectre and Meltdown

- Information flow
- Temporal relationships
- Multiple traces

Specification Mining

Specification Mining

Ernst et al. (2007) - Daikon

Hangal and Lam (2002) - Diduce

Hangal et al. (2005) - IODINE

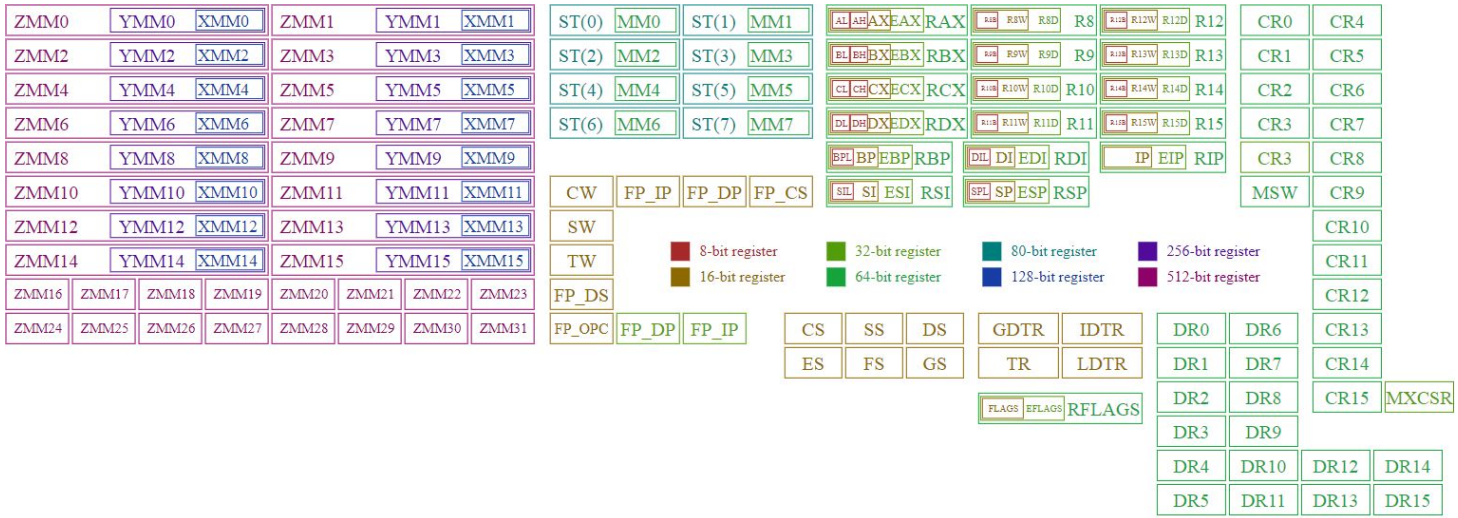
Lemieux et al. (2015) - Texada

Zhang et al. (2017) - SCIFinder

Smith et al. (2017) - Bach

Specification Mining (for Hardware Security)

We routinely wish to study systems for which no specification exists.



Common Question: Mining unverified systems

“Dynamic invariants… may be unsound; a false invariant is nevertheless useful, because it points to functionality not covered by the test-suite.”

-Hangal et al. (2005)

“Mined specifications cannot replace manually written specification... Nevertheless, as many programs lack formal specifications, mined specifications are valuable.”

-Lemieux et al. (2015)

Miners and Logics *taking Daikon as a baseline*

Spectre and Meltdown need a security, hardware, temporal, hyperproperty miner.

Daikon is a functional, software, invariant miner.

Can be **offline** or **online**.

Daikon v. Diduce

Can be over **invariants** or **temporal properties**.

Daikon v. Texada

Overview of miners

Can be for **software** or **hardware**.

Daikon v. Iodine

Can be over **functional** or **security** properties.

Daikon v. SCIFinder

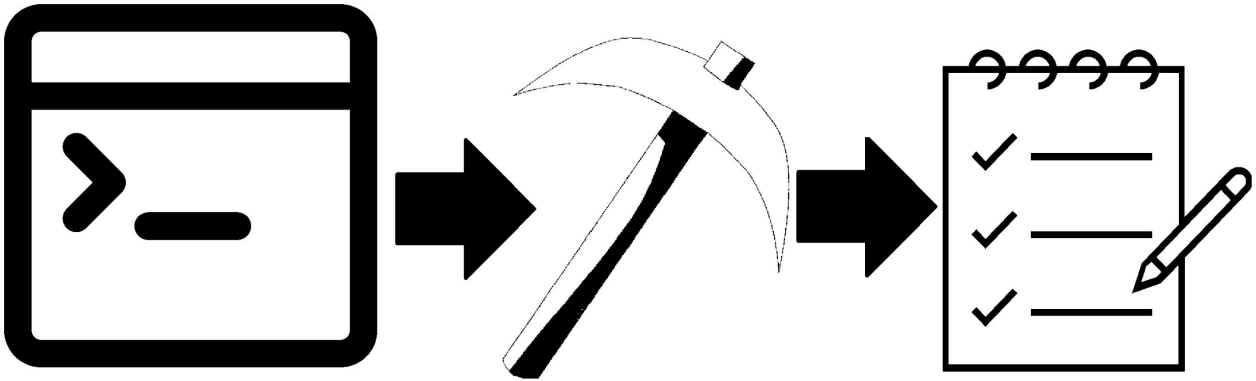
Can be over **trace properties** or **hyperproperties**.

Daikon v. Bach

Mining High Level Sketch

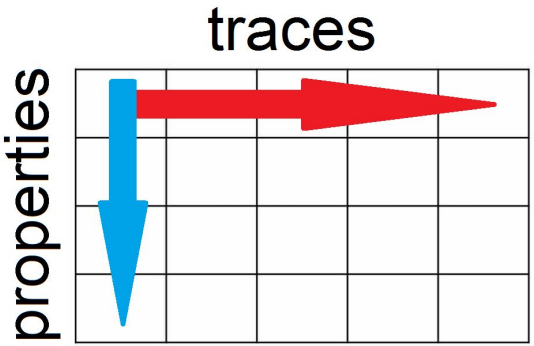
Miners often can find specifications manual work could not.

Miners often require manual assistance and testing suites.



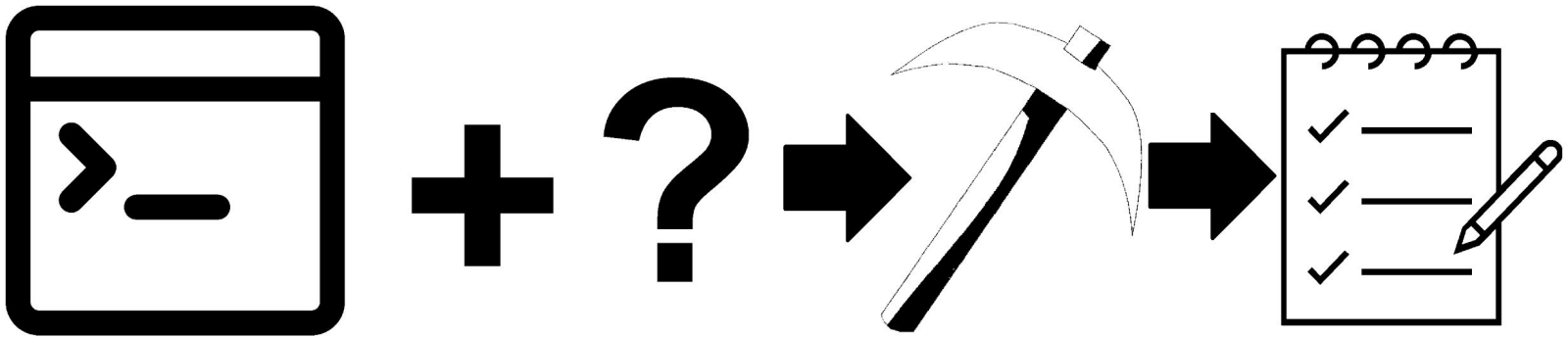
Methods: Mining Direction

- **Daikon, Diduce, Iodine** mine along a trace.
- **Texada** validates each property across the trace set.
- **Bach** mines relational specifications across function invocations.
- **SCIFinder** takes properties through an ML tagging process.



Methods: Inputs

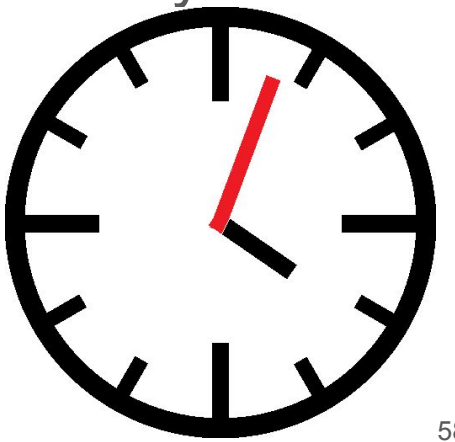
- **Daikon, Iodine** accept as input certain types of invariants to check
- **Texada** takes an LTL template.
- **SCIFinder** takes tagged properties.



Challenges: Complexity on Software

- **Diduce** incurs a 10-100x slowdown on examined software.
- **Daikon** is “usable at scale,” but with pared down features.
- **Bach** runs in minutes given an input Python library

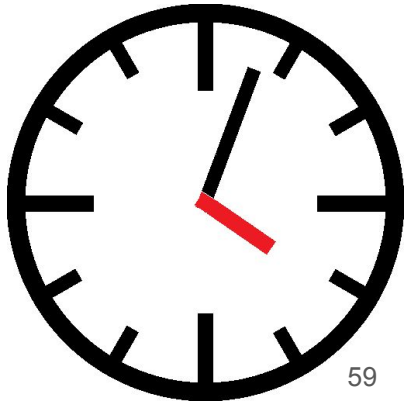
Generally software designs take minutes.



Challenges: Complexity on Hardware

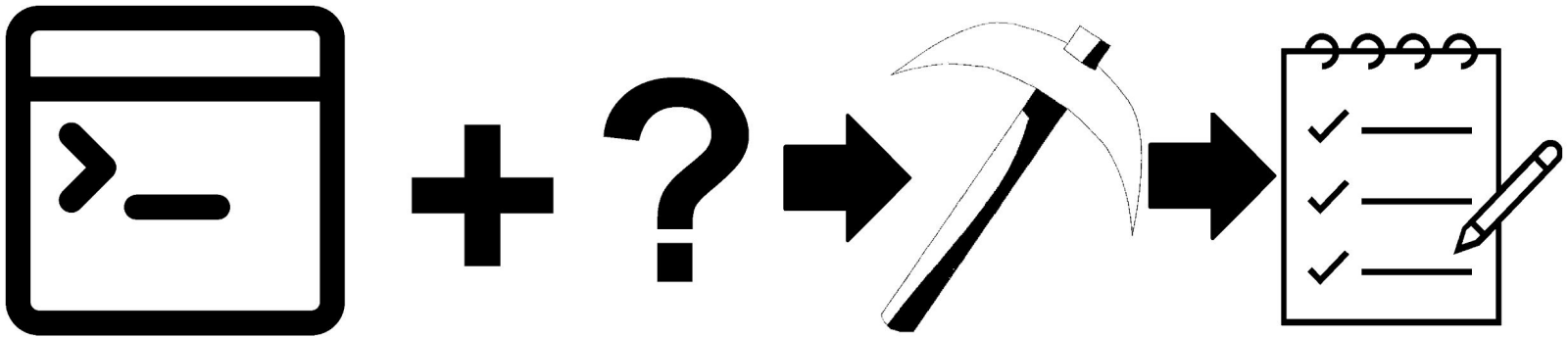
- **Daikon** runs in approximately 30 hours over a design in practice.
- **Texada** runs in tens of minutes over a design and template
- **SCIFinder** took 12 hours for 26 GB of trace data over a design.

Generally hardware designs take 10s of hours for trace properties.



Challenge: Beating Manual Methods

- **Diduce** uses the principle of “serendipitous invariants”
- **SCIFinder** and **Iodine** apply manual results.
- **Bach** exploits SMT engines and sql functions.
- **Daikon** follows software engineering practices, allows user input.

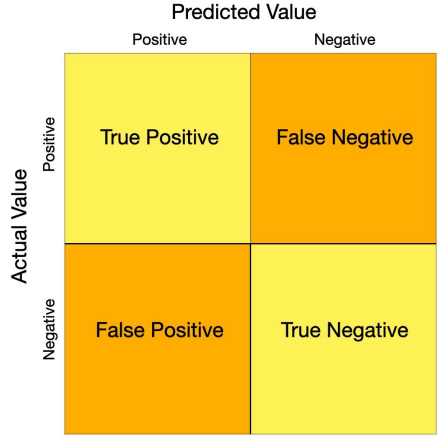


Challenge: Manageable Numbers of Properties

- **Texada** requires “property templates”
- **SCIFinder** uses ML generated security tags.
- **Iodine** and **Bach** only consider a subset, either one-hot, mutex, etc. or *relational specifications.*

Challenge: False Positives

- **Diduce** struggles with early invariant changes
- **Daikon, SCIFinder, Texada** vary their confidence levels.
- **Iodine** also wants false positives as outputs.
- **Bach** discovers an incorrect specification 10% of the time



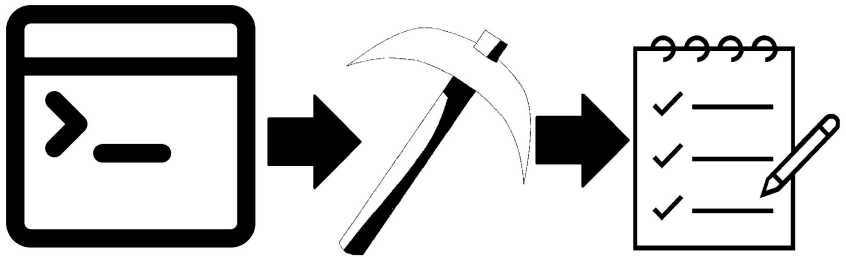
Miners, summary

Faster on software, slower on hardware.

Faster on invariants, slower on others.

Require manual assistance.

Outputs may be false or incomplete, but generally good.



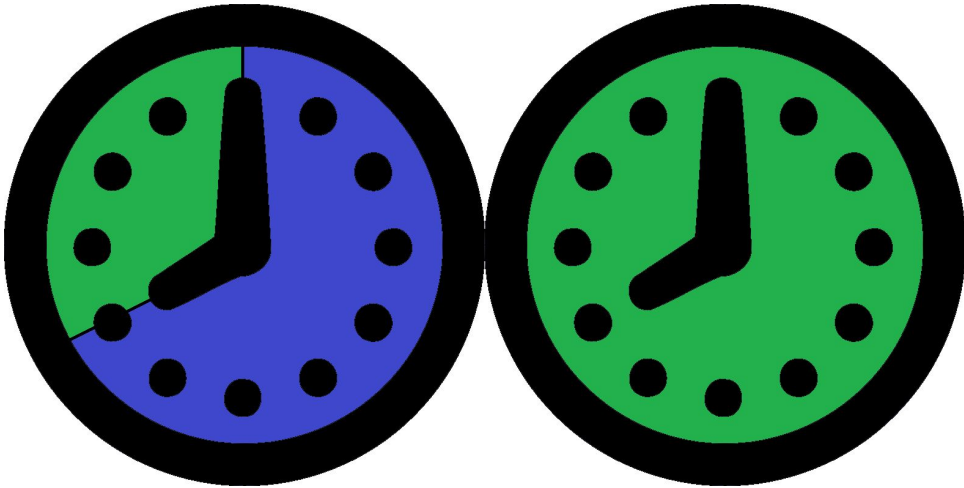
Format

2 hour examination

40-50 minute presentation

10 papers:

- 4 on logics of specifications
- 6 on specification mining



Denning (1976)

Alpern & Schneider (1987)

Clarkson & Schneider (2008)

Clarkson et al. (2014)

Ernst et al. (2007)

Hangal & Lam (2002)

Hangal et al. (2005)

Lemieux et al. (2015)

Zhang et al. (2017)

Smith et al. (2017)