Building Verified Program Analyzers in Coq

Lecture 1: Motivations and Examples

David Pichardie - INRIA Rennes / Harvard University

How do you trust your software ?

How do you trust your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

How do you trust your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

- Manual verifications
  - do not scale

manual verification

yesterday

How do you trust your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

- Manual verifications
  - do not scale
- Automatic bug finders
  - may miss some bugs

manual verification bug finders

yesterday today

How do you trust your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

- Manual verifications
  - do not scale
- Automatic bug finders
  - may miss some bugs
- Automatic, sound verifiers
  - find all bugs, may raise false alarms

  ex: the Astrée static analyzer

http://www.astree.ens.fr/

~1M loc of a critical control-command software analyzed

0 false alarms

manual verification	bug finders	sound verifiers
yesterday	today	tomorrow

How do you trust the tool that verifies your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

- Manual verifications
  - do not scale
- Automatic bug finders
  - may miss some bugs
- Automatic, sound verifiers
  - find all bugs, may raise false alarms

  ex: the Astrée static analyzer

http://www.astree.ens.fr/

~1M loc of a critical control-command software analyzed

0 false alarms

manual verification	bug finders	sound verifiers
yesterday	today	tomorrow

How do you trust the tool that verifies your software ?

The increasing complexity of safety critical systems requires efficient validation techniques

- Manual verifications
  - do not scale
- Automatic bug finders
  - may miss some bugs
- Automatic, sound verifiers
  - find all bugs, may raise false alarms

  ex: the Astrée static analyzer
- Formally-verified verifiers
  - the verifier comes with a soundness proof
  - that is machine checked

http://www.astree.ens.fr/

~1M loc of a critical control-command software analyzed

0 false alarms

manual verification	bug finders	sound verifiers	verified verifiers
yesterday	today	tomorrow	after tomorrow

How do we verify the verifier?

How do we verify the verifier?

A simple idea:

How do we verify the verifier?

A simple idea:

Program and prove your verifier in the same language!

How do we verify the verifier?

A simple idea:

Program and prove your verifier in the same language!

Which language ?

How do we verify the verifier?

A simple idea:

Program and prove your verifier in the same language!

Which language ?

Coq

Coq: an animal with two faces

Coq: an animal with two faces

First face:

Coq: an animal with two faces

First face:

- a proof assistant that allows to interactively build proof in constructive logic

Coq: an animal with two faces

First face:

- a proof assistant that allows to interactively build proof in constructive logic

Second face:

Coq: an animal with two faces

First face:

- a proof assistant that allows to interactively build proof in constructive logic

Second face:

- a functional programming language with a very rich type system

Coq: an animal with two faces

First face:

- a proof assistant that allows to interactively build proof in constructive logic

Second face:

- a functional programming language with a very rich type system

  example:

```
sort: ∀ l: list int, { l': list int |
                            Sorted l' ∧ PermutationOf l l' }
```

Coq: an animal with two faces

First face:

- a proof assistant that allows to interactively build proof in constructive logic

Second face:

- a functional programming language with a very rich type system

  example:

  ```
  sort: ∀ l: list int, { l’: list int |
                                Sorted l’ ∧ PermutationOf l l’ }
  ```

- with an extraction mechanism to Ocaml

  ```
  sort: int list → int list
  ```

Coming soon...

The next lecture will provide a short introduction to Coq

You may want to install the tool on you computer during this first lecture (but please, try to keep focused nevertheless...)

Instructions for installation:

http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

Coming soon...

The next lecture will provide a short introduction to Coq

You may want to install the tool on you computer during this first lecture (but please, try to keep focused nevertheless...)

Instructions for installation:

http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

Our Methodology

Our Methodology

We program the static analyzer inside Coq

```
Definition analyzer (p:program) := ...
```

Static Analyzer

Logical Framework
(here Coq)

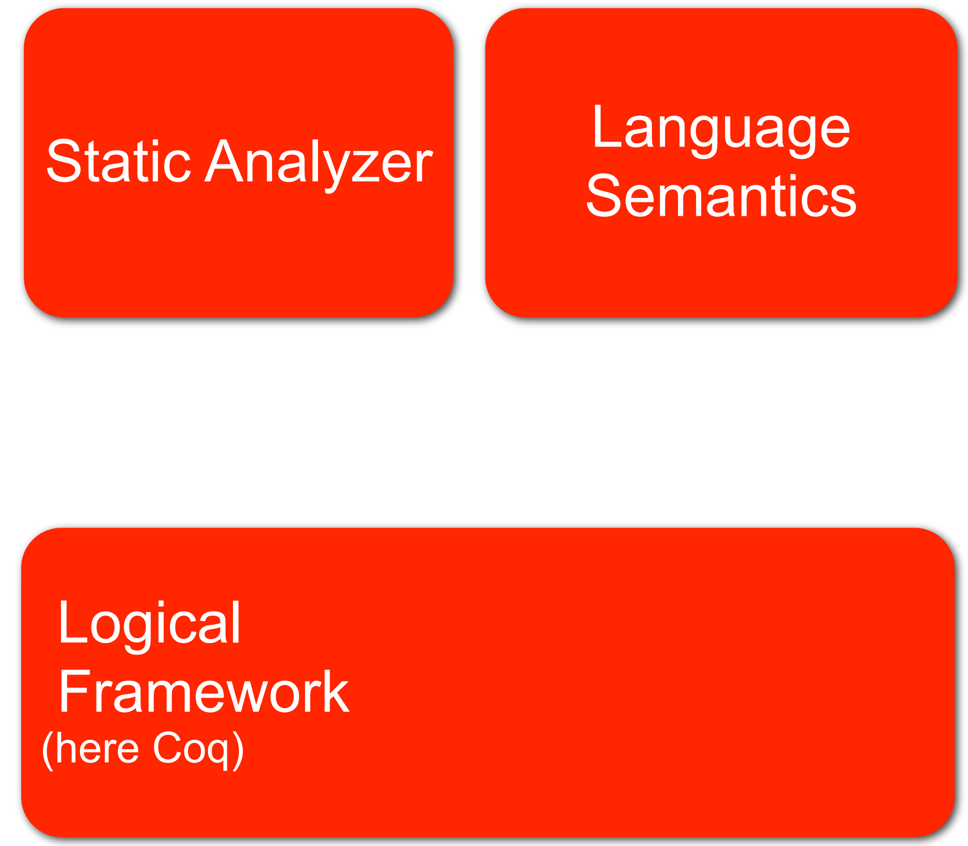
Our Methodology

We program the static analyzer inside Coq

```
Definition analyzer (p:program) := ...
```

We state its correctness wrt. a formal specification of the language semantics

```
Theorem analyser_is_sound :
  ∀ p, analyser p = Yes → Sound(p)
```



Our Methodology

We program the static analyzer inside Coq

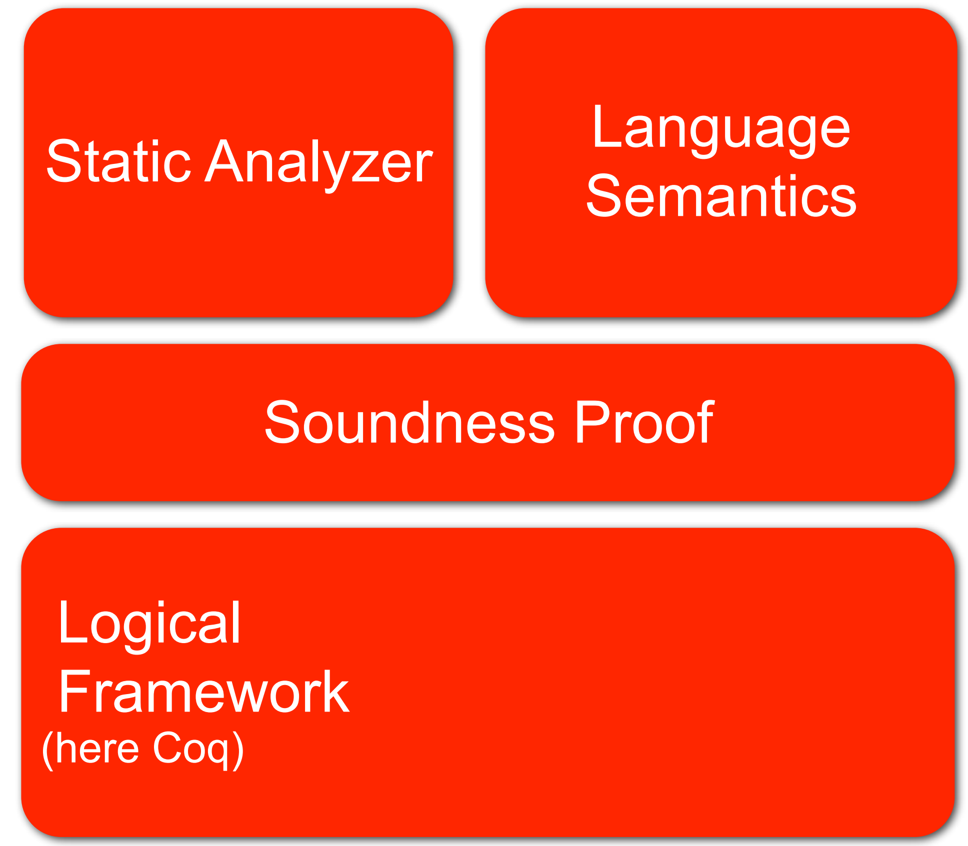
```
Definition analyzer (p:program) := ...
```

We state its correctness wrt. a formal specification of the language semantics

```
Theorem analyser_is_sound :
  ∀ p, analyser p = Yes → Sound(p)
```

We interactively and mechanically prove this theorem

```
Proof. ... (* few days later *) ... Qed.
```



http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

Our Methodology

We program the static analyzer inside Coq

```
Definition analyzer (p:program) := ...
```

We state its correctness wrt. a formal specification of the language semantics

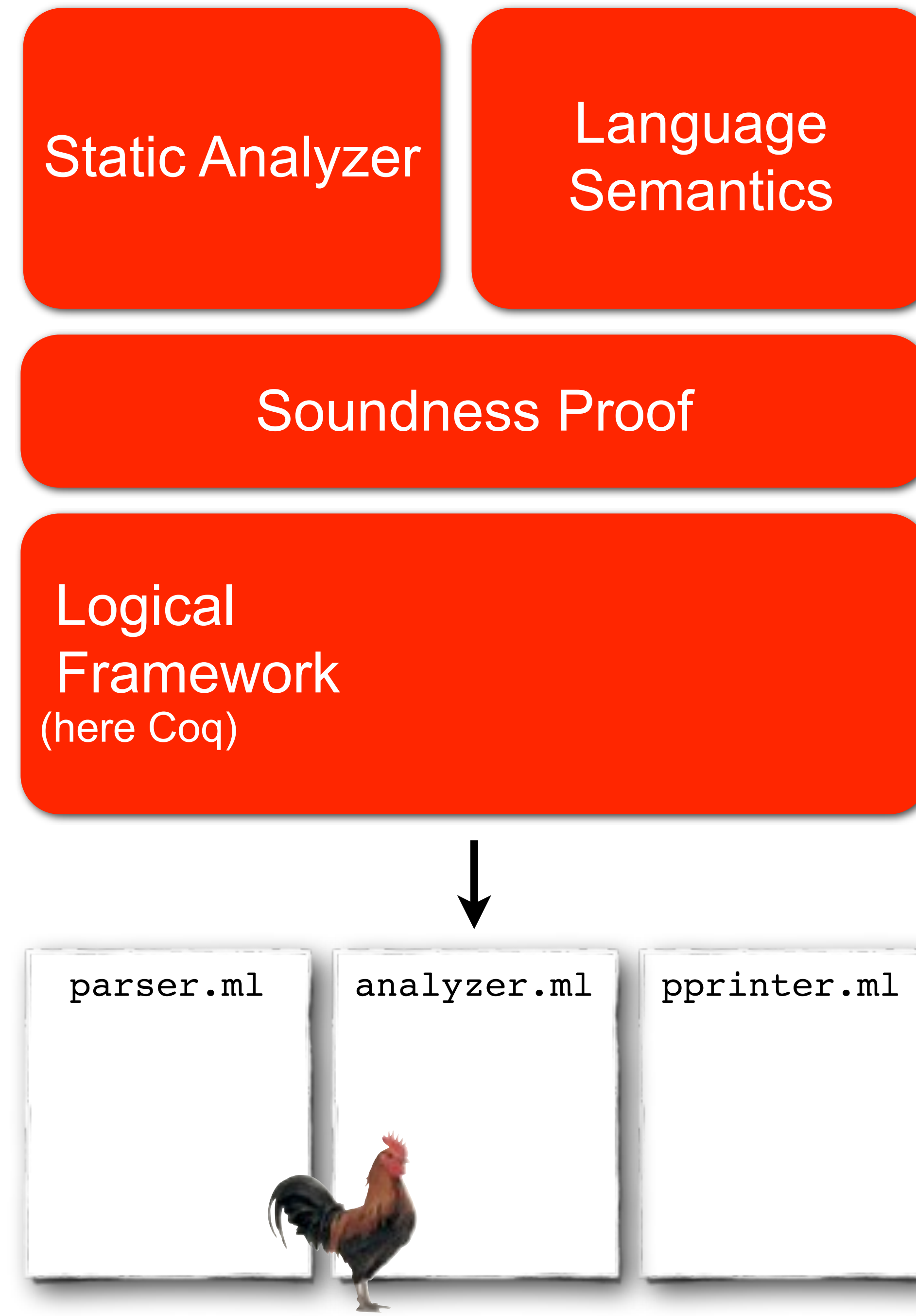
```
Theorem analyser_is_sound :
  ∀ p, analyser p = Yes → Sound(p)
```

We interactively and mechanically prove this theorem

```
Proof. ... (* few days later *) ... Qed.
```

We extract an OCaml implementation of the analyzer

```
Extraction analyzer.
```



http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

A Posteriori Validation

An important tool in our toolbox

We program the *full* static analyzer inside Coq

```
Definition analyzer (p:program) :=
  ...
  let x := complex_computation p in
  ...
```

... or ...

A Posteriori Validation

An important tool in our toolbox

We program the *full* static analyzer inside Coq

```
Definition analyzer (p:program) :=
  ...
  let x := complex_computation p in
  ...
```

... or we program some parts in Coq, other parts in OCaml and use a verified validator

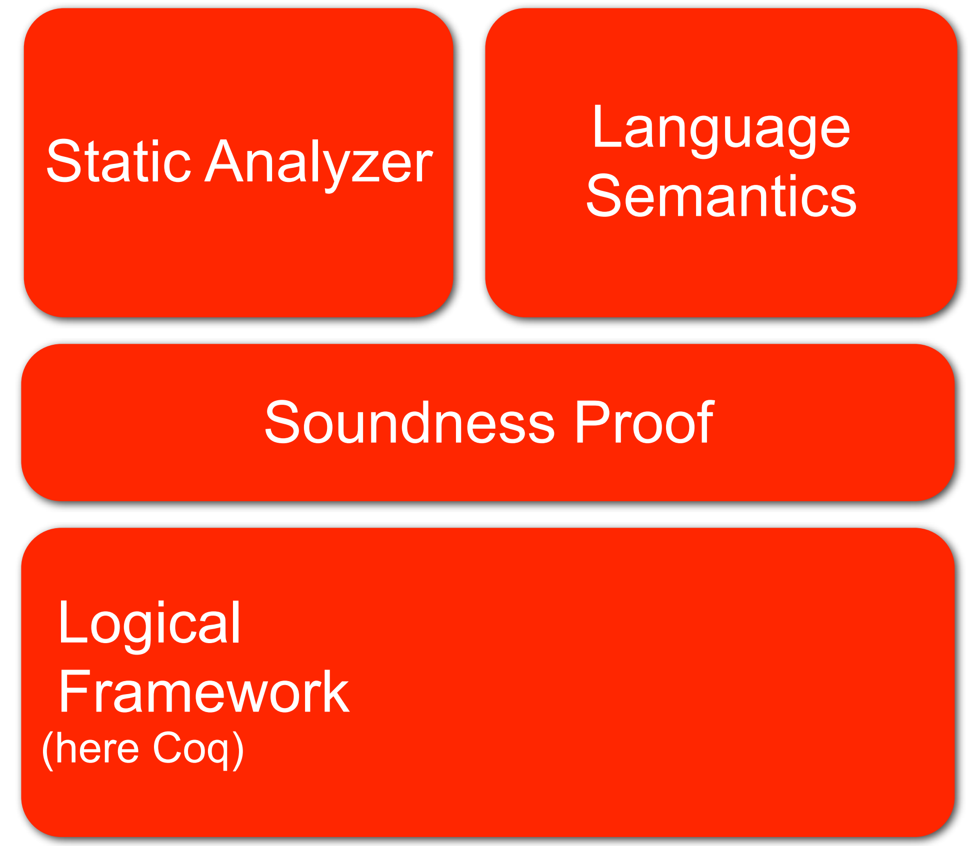
```
Definition analyzer (p:program) :=
  ...
  let x := ocaml_external_complex_computation p in
  match validator x p with
    | OK ⇒ ...
    | Error ⇒ abort
  end.
```

Ideally we also prove (on paper) that if the external implementation implements correctly a well-known algorithm then the validator will always succeed (completeness)

Trusted Computing Base (TCB)

1. Formal specification of the programming language semantics
   - (informally) shared by any end-user programmer, compiler, static analyzer
   - less specialized than static analyzer’s abstract semantics

2. Logical Framework
   - only the proof checker needs to be trusted
   - we don’t trust sophisticated decision procedures

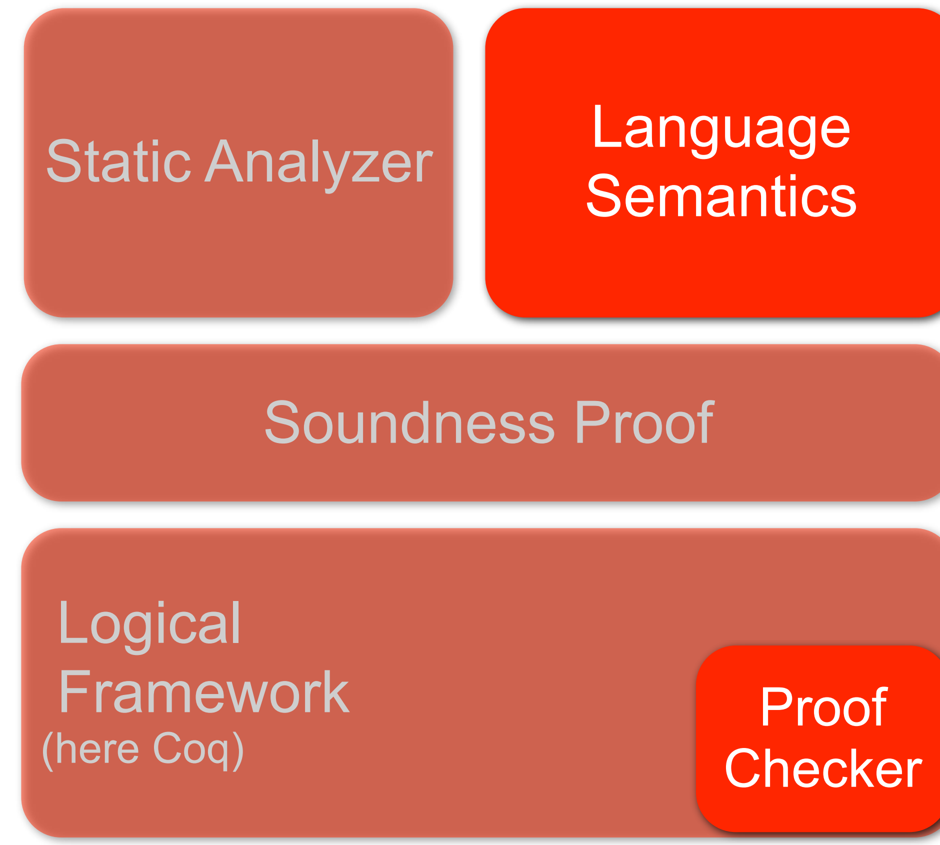


http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

Trusted Computing Base (TCB)

1. Formal specification of the programming language semantics
   - (informally) shared by any end-user programmer, compiler, static analyzer
   - less specialized than static analyzer's abstract semantics

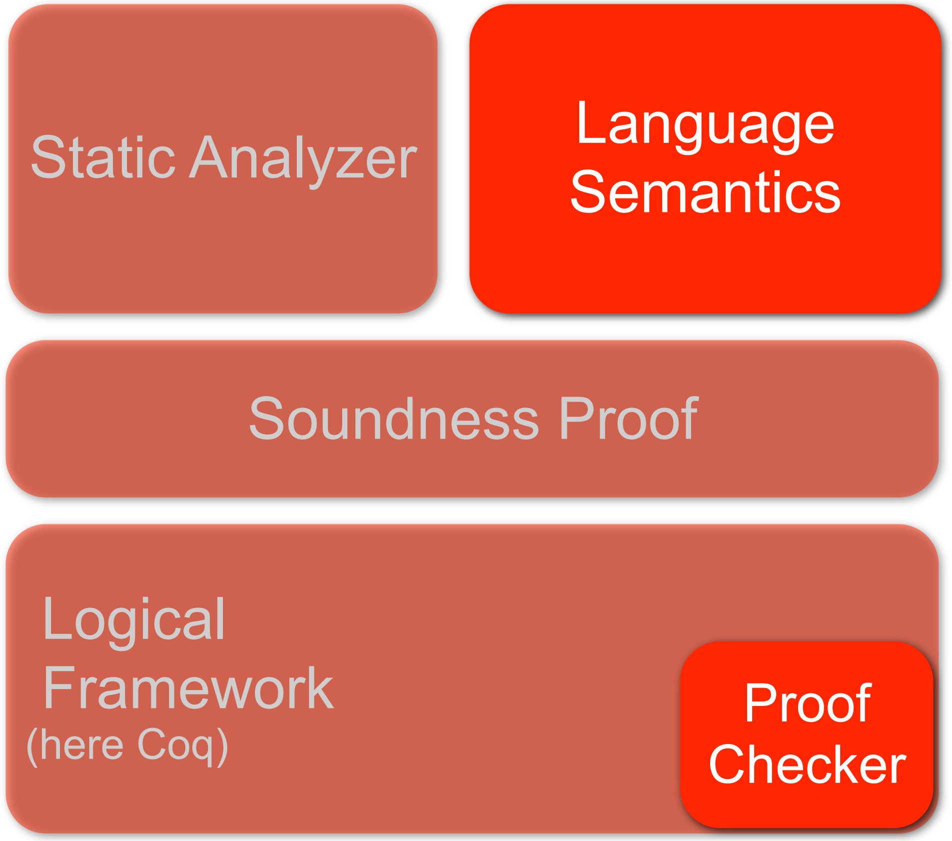
2. Logical Framework
   - only the proof checker needs to be trusted
   - we don't trust sophisticated decision procedures



Trusted Computing Base (TCB)

1. Formal specification of the programming language semantics
   - (informally) shared by any end-user programmer, compiler, static analyzer
   - less specialized than static analyzer's abstract semantics

2. Logical Framework
   - only the proof checker needs to be trusted
   - we don't trust sophisticated decision procedures



Still a large code base but at least a *foundational* code base: logic & semantics

http://www.irisa.fr/celtique/pichardie/teaching/digicosme13

Using Proof Assistants in PL Research

We distinguish two approaches

1. Accompany a research paper with a machine checked proof
   - increase the trust in a scientific result
   - facilitate the review process

2. Help building highly reliable *meta-programs* (programs that manipulate programs: compilers, program verifiers, type checkers...)
   - require to program the tool inside the proof assistant (generic interface, efficient data-structures)
   - must scale to realistic languages with large formal semantics

Verified PL Stacks: Achievements

Some major achievements have changed our expectations about programming language mechanized proofs

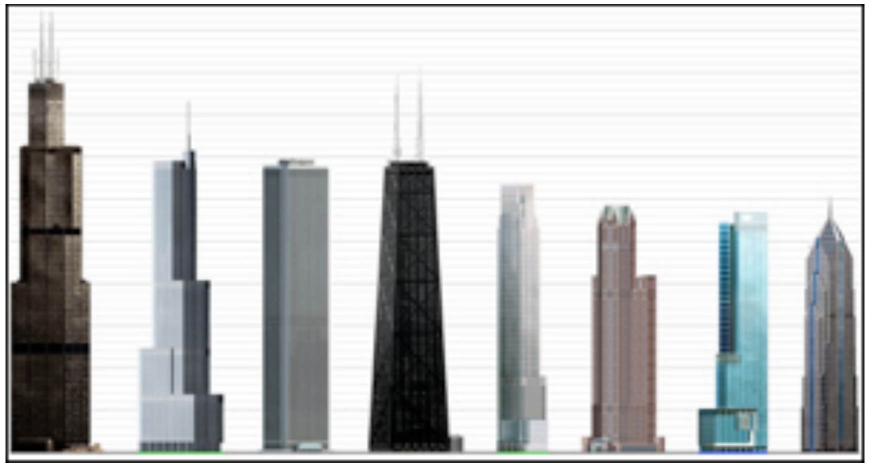
- M6: JVM bytecode interpreter in ACL2 (Liu)
- Jinja: source & bytecode Java, compiler, BCV in Isabelle/HOL (Klein & Nipkow, and extensions by Lochbihler)
- CompCert: realistic C compiler in Coq (Leroy, Blazy et al.)
- Verified Software ToolChain: extension of CompCert for concurrent C programs (Appel et al.)
- seL4: verified OS kernel in Isabelle/HOL (Klein, Norrish et al.)

Verified Static Analysis: Objectives

We identify two major objectives

1. building new proof methods for the working (mechanized) semanticist
    - using Abstract Interpretation theory, we can provide generic interfaces between analyses
    - we have to discover new *a posteriori* validation algorithms (sound and efficient)

2. building big proofs
    - proving in the small will not give us all the lessons we want to learn
    - large case studies are important to build a new *proof engineering* knowledge



These Lectures

Lecture 1

Motivations

Examples of verified analysers

Lecture 2

Coq crash course

Lecture 3

Verified abstract interpreter for a simple imperative language

Lecture 4

CompCert

A verified value analysis for CompCert

Some example of verified static analysers

Lecture objectives

- Show examples of challenging verified static analysis
- Complement your knowledge on static analysis

Two examples

- Information flow type system
- data-race static analysis

These analyses target the Java (bytecode) language

A Certified Non-Interference Java Bytecode Verifier

G. Barthe, D. Pichardie and T. Rezk, A Certified Lightweight Non-Interference Java Bytecode Verifier, ESOP'07

Motivations: Bytecode Verification

Java bytecode verification

- checks that applets are correctly formed and correctly typed,
- using a static analysis of bytecode programs

But Java bytecode verifier (and more generally Java security model)

- only concentrates on who accesses sensitive information,
- not how sensitive information flows through programs

In this work

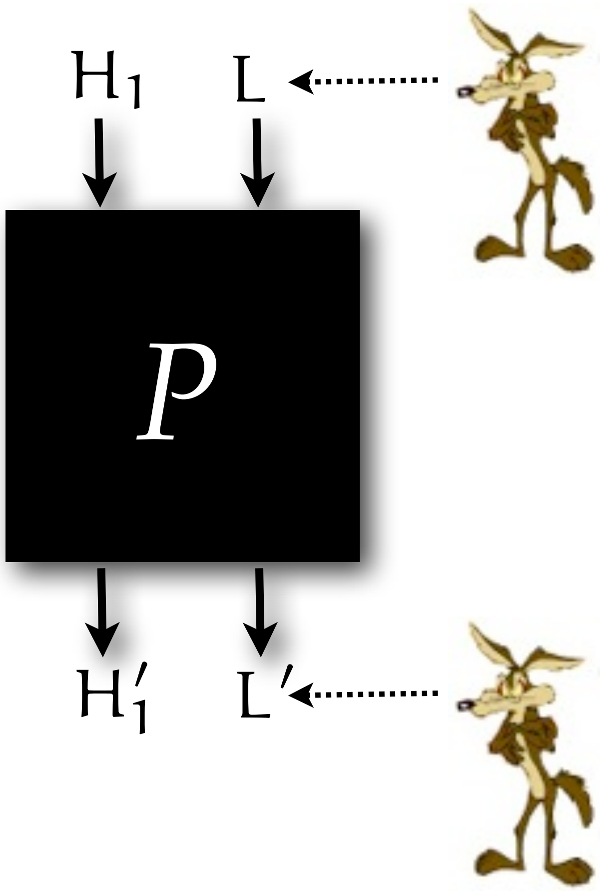
- We propose an information flow type system for a sequential JVM-like language, including classes, objects, arrays, exceptions and method calls.
- We prove in Coq that it guarantees the semantical non-interference property on method input/output.

Non-Interference

"Low-security behavior of the program is not affected by any high-security data." *Goguen&Meseguer 1982*

High = secret

Low = public

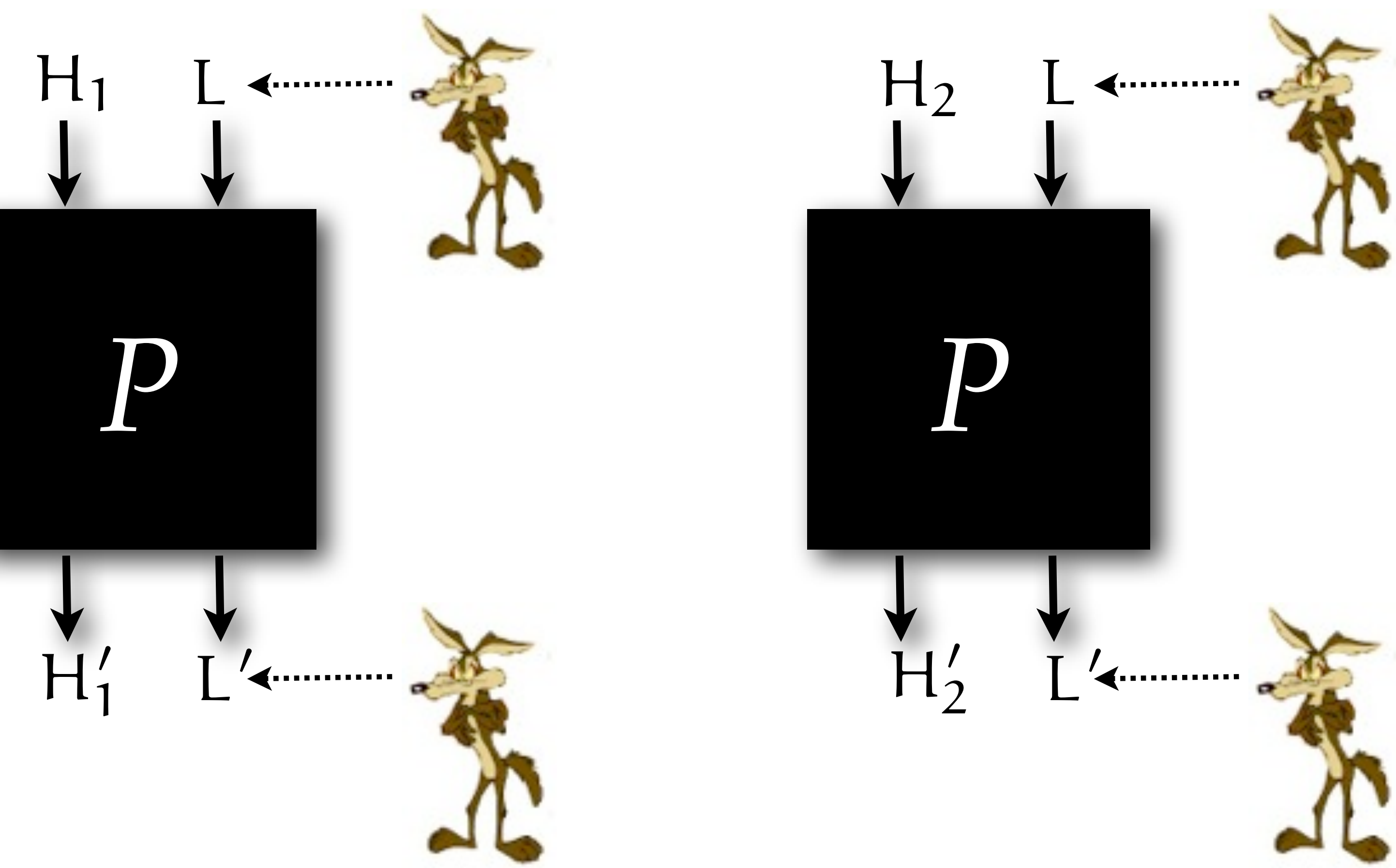


Non-Interference

"Low-security behavior of the program is not affected by any high-security data." *Goguen&Meseguer 1982*

High = secret

Low = public

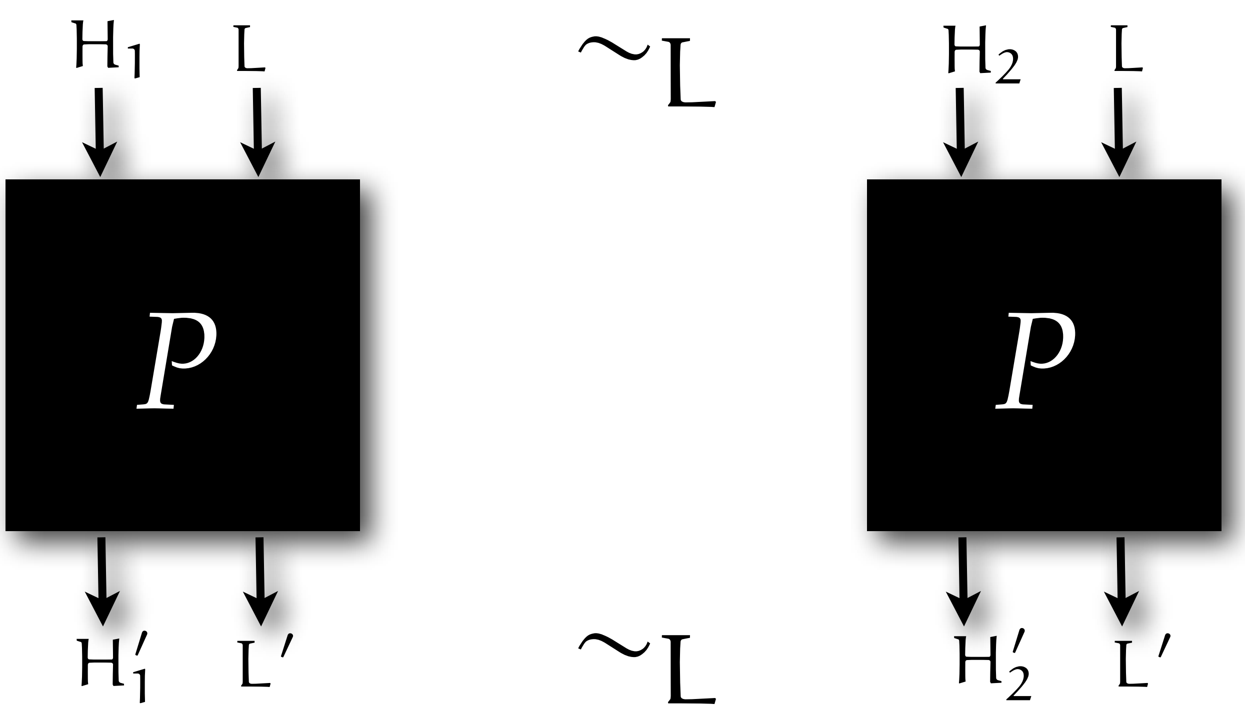


Non-Interference

"Low-security behavior of the program is not affected by any high-security data." *Goguen&Meseguer 1982*

High = secret

Low = public



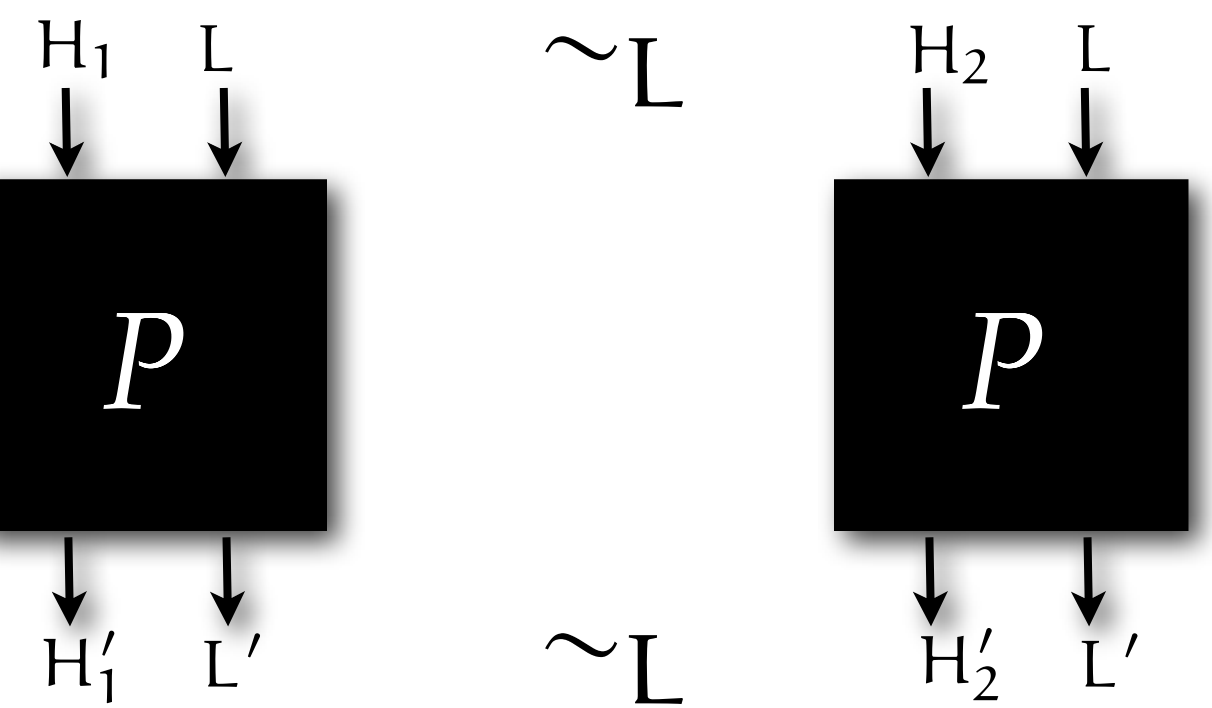
$$\forall s_1\ s_2,\ s_1 \sim_L s_2 \Longrightarrow [\![P]\!](s_1) \sim_L [\![P]\!](s_2)$$

Non-Interference

"Low-security behavior of the program is not affected by any high-security data." *Goguen&Meseguer 1982*

High = secret

Low = public



$$\forall s_1\ s_2,\ s_1 \sim_L s_2 \Longrightarrow [\![P]\!](s_1) \sim_L [\![P]\!](s_2)$$

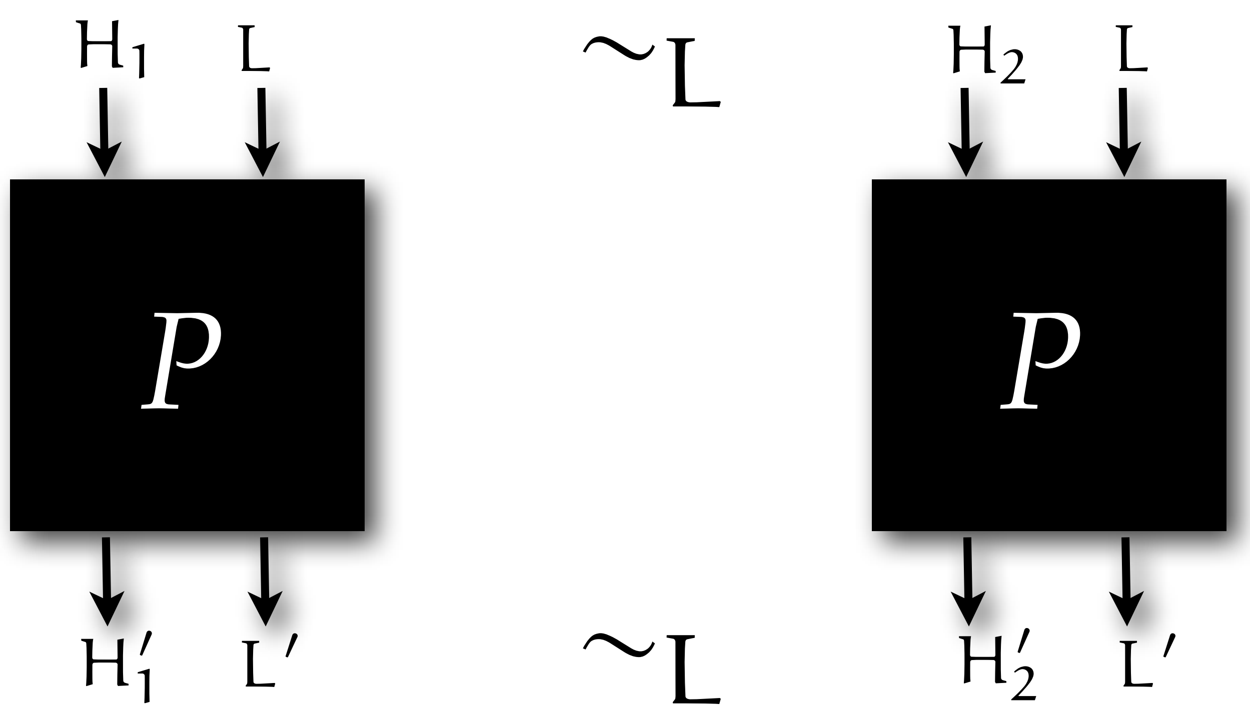
if inputs are equivalent for the attacker...

Non-Interference

"Low-security behavior of the program is not affected by any high-security data." *Goguen&Meseguer 1982*

High = secret

Low = public



$$\forall s_1\ s_2,\ s_1 \sim_L s_2 \Longrightarrow [\![P]\!](s_1) \sim_L [\![P]\!](s_2)$$

if inputs are equivalent for the attacker...

...outputs should be equivalent too.

Example of information leaks

Explicit flow:

```
public int{L} foo(int{L} l; int{H} h) {
  return h;
}
```

Implicit flow:

```
 public int{L} foo(int{L} l1; int{L} l2; int{H} h) {
 if (h==0) {return l1;} else {return l2;};
}
```

We use here the Jif (http://www.cs.cornell.edu/jif) syntax:

- a security-typed extension of Java (source) with support for information flow.

Information flow type systems

Previous work

- Non-interference can be enforced by a type system [Volpano97]
- The type system is sound: it rejects every interferent programs

$$WellTyped(P) \implies NonInterferent(P)$$

Information flow type systems

Previous work

- Non-interference can be enforced by a type system [Volpano97]
- The type system is sound: it rejects every interferent programs

$$WellTyped(P) \implies NonInterferent(P)$$

D. Volpano and G. Smith, *A Type-Based Approach to Program Security*, Theory and Practice of Software Development, 1997.

$$\text{CONST}\ \frac{}{\vdash n : L} \qquad \text{VAR}\ \frac{x \in \mathbb{V}_\tau}{\vdash x : \tau} \qquad \text{BINOP}\ \frac{\vdash e_1 : \tau \qquad \vdash e_2 : \tau}{\vdash e_1\ o\ e_2 : \tau}$$

$$\text{EXP-SUBTYP}\ \frac{\vdash e : \tau_1 \qquad \tau_1 \sqsubseteq \tau_2}{\vdash e : \tau_2}$$

$$\text{ASSIGN}\ \frac{x \in \mathbb{V}_\tau \qquad \vdash e : \tau}{\vdash x := e : \tau} \qquad \text{SEQ}\ \frac{\vdash S_1 : \tau \qquad \vdash S_2 : \tau}{\vdash S_1\ ;\ S_2 : \tau}$$

$$\text{IF}\ \frac{\vdash e : \tau \qquad \vdash S_1 : \tau \qquad \vdash S_2 : \tau}{\vdash \texttt{if}\ e\ \texttt{then}\ S_1\ \texttt{else}\ S_2 : \tau} \qquad \text{WHILE}\ \frac{\vdash e : \tau \qquad \vdash S : \tau}{\vdash \texttt{while}\ e\ \texttt{do}\ S : \tau}$$

$$\text{STM-SUBTYP}\ \frac{\vdash S : \tau_2 \qquad \tau_1 \sqsubseteq \tau_2}{\vdash S : \tau_1}$$

Information flow type systems

Previous work

- Non-interference can be enforced by a type system [Volpano97]
- The type system is sound: it rejects every interferent programs

$$WellTyped(P) \implies NonInterferent(P)$$

Achievements: mechanized proof of a type-checker for a bytecode langage handling

- unstructured control flow
- operand stack
- exceptions
- objects and array dynamically allocated
- classes and virtual method calls

(Preliminary) experiments:

- we have extracted the type ckecker and used it on a small case study

```
| vaload_np_caught : forall i te t k1 k2 ke st,
  (forall j, region i (Some np) j -> k2 <= se j) ->
  handler i np = Some te ->
  texec i (Vaload t) (Some np) (L.Simple k1::L.Array k2 ke::st) (Some (L.Simple k2::nil))
| vaload_np_uncaught : forall i t k1 k2 ke st,
  (forall j, region i (Some np) j -> k2 <= se j) ->
  k2 <= sgn.(resExceptionType) np ->
  handler i np = None ->
  texec i (Vaload t) (Some np) (L.Simple k1::L.Array k2 ke::st) None
| vaload_iob_caught : forall i te t k1 k2 ke st,
  (forall j, region i (Some iob) j -> k1 U k2 <= se j) ->
  handler i iob = Some te ->
  texec i (Vaload t) (Some iob) (L.Simple k1::L.Array k2 ke::st) (Some (L.Simple (k1 U k2)
| vaload_iob_uncaught : forall i t k1 k2 ke st,
  (forall j, region i (Some iob) j -> k1 U k2 <= se j) ->
  k1 U k2 <= sgn.(resExceptionType) iob ->
  handler i iob = None ->
  texec i (Vaload t) (Some iob) (L.Simple k1::L.Array k2 ke::st) None
| vastore : forall i t kv ki ka ke st,
  kv <=' ke ->
  ki <= ke ->
  ka <= ke ->
  (forall j, region i None j -> ka <= se j) ->
  (forall j, region i None j -> (L.join ki ka) <= (se j)) ->
  (forall j, region i None j -> ke <= (se j)) ->
  L.leql' (heapEffect sgn) ke ->
  texec i (Vastore t) None (kv::L.Simple ki::L.Array ka ke::st) (Some (elift m i ke st))
| vastore_np_caught : forall i te t kv ki ka ke st,
  (forall j, region i (Some np) j -> ka <= se j) ->
  handler i np = Some te ->
  texec i (Vastore t) (Some np) (kv::L.Simple ki::L.Array ka ke::st) (Some (L.Simple ka::n
| vastore_np_uncaught : forall i t kv ki ka ke st,
  (forall j, region i (Some np) j -> ka <= se j) ->
  ka <= sgn.(resExceptionType) np ->
  handler i np = None ->
  texec i (Vastore t) (Some np) (kv::L.Simple ki::L.Array ka ke::st) None
| vastore_ase_caught : forall i te t ki ka (kv ke:L.t') st,
  (forall j, region i (Some ase) j -> (L.join kv (L.join ki ka)) <= se j) ->
  handler i ase = Some te ->
  texec i (Vastore t) (Some ase) (kv::L.Simple ki::L.Array ka ke::st)
                                      (Some (L.Simple (L.join kv (L.join ki ka))::nil))
| vastore_ase_uncaught : forall i t ki ka (kv ke:L.t') st,
  (forall j, region i (Some ase) j -> (L.join kv (L.join ki ka)) <= se j) ->
  (L.join kv (L.join ki ka)) <= sgn.(resExceptionType) ase ->
  handler i ase = None ->
  texec i (Vastore t) (Some ase) (kv::L.Simple ki::L.Array ka ke::st) None
| vastore_iob_caught : forall i te t ki ka (kv ke:L.t') st,
  (forall j, region i (Some iob) j -> (L.join ki ka) <= se j) ->
  handler i iob = Some te ->
  texec i (Vastore t) (Some iob) (kv::L.Simple ki::L.Array ka ke::st) (Some (L.Simple (L.j
| vastore_iob_uncaught : forall i t ki ka (kv ke:L.t') st,
  (forall j, region i (Some iob) j -> (L.join ki ka) <= se j) ->
  (L.join ki ka) <= sgn.(resExceptionType) iob ->
  handler i iob = None ->
  texec i (Vastore t) (Some iob) (kv::L.Simple ki::L.Array ka ke::st) None
| vload : forall i t x st,
  texec i (Vload t x) None st (Some (L.join' (se i) (sgn.(lvt) x)::st))
| vstore : forall i t x k st,
  se i <= sgn.(lvt) x ->
  L.leql' k (sgn.(lvt) x) ->
  texec i (Vstore t x) None (k::st) (Some st)
| vreturn : forall i x k kv st,
  sgn.(resType) = Some kv ->
  L.leql' k kv ->
  texec i (Vreturn x) None (k::st) None.
```

66 typing rules...

Some reading

A. Myers. *Expressing and Enforcing Security with Programming Languages.* PLDI'06 tutorial.

A. Sabelfeld and A. Myers. *Language-Based Information-Flow Security.* IEEE Journal on Selected Areas in Communication, 2003

D. Volpano and G. Smith. *A Type-Based Approach to Program Security.* Theory and Practice of Software Development, 1997

A. Sabelfeld and D.Sands. *Declassification: Dimensions and principles.* Journal of Computer Security, 2009.

T. H. Austin and C. Flanagan. *Efficient purely-dynamic information flow analysis.* PLAS 2009

A Certified Data Race Analysis for Java Bytecode

F. Dabrowski and D. Pichardie, A Certified Data Race Analysis for a Java-like Language, TPHOLs 2009

Data Races

A fundamental issue in multi-threaded programming

Definition: *the situation where two different processes attempt to access to the same memory location and at least one access is a write.*

Leads to tricky bugs

- difficult to reproduce and identify via manual code review or program testing

Wanted: sequentially consistent (SC) executions

- each thread accesses instantly a common shared memory
- the execution can be modelled with an interleaving of thread actions

The Java specification gives very surprising semantics to programs with races

- Only data-race-free programs are guaranteed to have only SC executions

Example

```
        C.f = C.g = 0;
1: x = C.g; ‖ 1: y = C.f;
2: C.f = 1; ‖ 2: C.g = 1;
```

x=1 and y=1
should be
impossible

Example

```
C.f = C.g = 0;
1: x = C.g; ‖ 1: y = C.f;
2: C.f = 1; ‖ 2: C.g = 1;
```

x=1 and y=1
should be
impossible

Interleaving semantics gives only sequentially consistent execution,

Example

```
C.f = C.g = 0;
1: x = C.g; ‖ 1: y = C.f;
2: C.f = 1; ‖ 2: C.g = 1;
```

x=1 and y=1
should be
impossible

Interleaving semantics gives only sequentially consistent execution,

```
1: x = C.g;
2: C.f = 1;
1: y = C.f;
2: C.g = 1;
```

Example

```
C.f = C.g = 0;
1: x = C.g; ‖ 1: y = C.f;
2: C.f = 1; ‖ 2: C.g = 1;
```

x=1 and y=1 should be impossible

Interleaving semantics gives only sequentially consistent execution,

```
1: x = C.g;      1: y = C.f;
2: C.f = 1;      2: C.g = 1;
1: y = C.f;      1: x = C.g;
2: C.g = 1;      2: C.f = 1;
```

Example

```
C.f = C.g = 0;
1: x = C.g; ║ 1: y = C.f;
2: C.f = 1; ║ 2: C.g = 1;
```

x=1 and y=1 should be impossible

Interleaving semantics gives only sequentially consistent execution,

```
1: x = C.g;      1: y = C.f;      1: y = C.f;
2: C.f = 1;      2: C.g = 1;      1: x = C.g;     ...
1: y = C.f;      1: x = C.g;      2: C.g = 1;
2: C.g = 1;      2: C.f = 1;      2: C.f = 1;
```

Example

x=1 and y=1
should be
impossible

```
C.f = C.g = 0;
1: x = C.g; || 1: y = C.f;
2: C.f = 1; || 2: C.g = 1;
```

Interleaving semantics gives only sequentially consistent execution,

```
1: x = C.g;    1: y = C.f;    1: y = C.f;         2: C.g = 1;
2: C.f = 1;    2: C.g = 1;    1: x = C.g;   ...   2: C.f = 1;
1: y = C.f;    1: x = C.g;    2: C.g = 1;         1: x = C.g;
2: C.g = 1;    2: C.f = 1;    2: C.f = 1;         1: y = C.f;
          ✔              ✔              ✔                   ✘
```

`x=1` and `y=1` is a legal outcome according to Java specification!

but such program may also lead to sequentially inconsistent execution

Origins:

- Multicore cache mechanisms
- Agressive compiler optimizations

Java Data-Race-Free Guarantee

The Java specification guarantees that if all SC execution of a program are race-free then the program will only exhibits these executions

A date-race verifier must infer automatically that a program is race free

- the verifier computes a set of potential races

$$\mathit{verifier}_{\mathrm{DR}} : \mathrm{program} \rightarrow \mathcal{P}(\mathrm{races})$$

- if this set is empty, the program must be free of races

$$\mathit{verifier}_{\mathrm{DR}}(P) = \emptyset \implies \mathit{Races}(P) = \emptyset$$

Java Data-Race-Free Guarantee

The Java specification guarantees that if all SC execution of a program are race-free then the program will only exhibits these executions

A date-race verifier must infer automatically that a program is race free

- the verifier computes a set of potential races

$$\mathit{verifier}_{\mathrm{DR}} : \mathrm{program} \to \mathcal{P}(\mathrm{races})$$

- if this set is empty, the program must be free of races

$$\mathit{verifier}_{\mathrm{DR}}(P) = \emptyset \implies \mathit{Races}(P) = \emptyset$$

computable

Java Data-Race-Free Guarantee

The Java specification guarantees that if all SC execution of a program are race-free then the program will only exhibits these executions

A date-race verifier must infer automatically that a program is race free

- the verifier computes a set of potential races

$$\mathit{verifier}_{\mathrm{DR}} : \mathrm{program} \rightarrow \mathcal{P}(\mathrm{races})$$

- if this set is empty, the program must be free of races

$$\mathit{verifier}_{\mathrm{DR}}(P) = \emptyset \implies \mathit{Races}(P) = \emptyset$$

computable

not-computable

Data Race Analysis

A Challenging Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Data Race Analysis

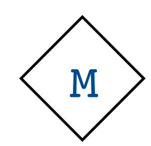
A Challenging Example

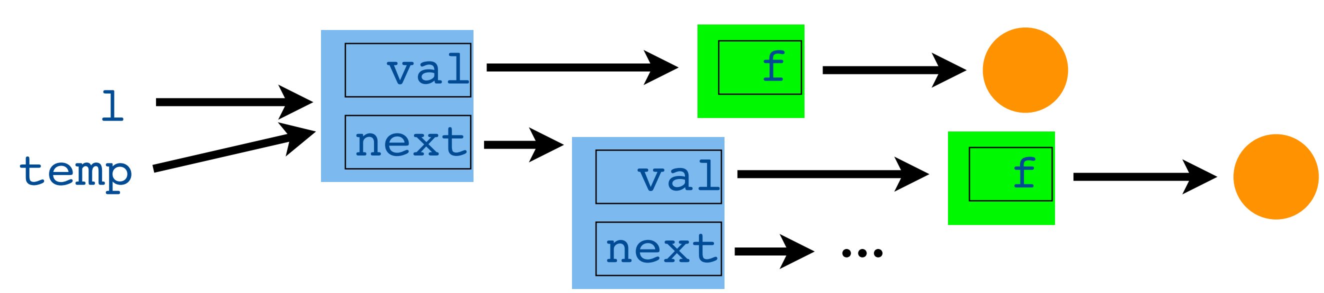
```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`

Threads: 



Data Race Analysis

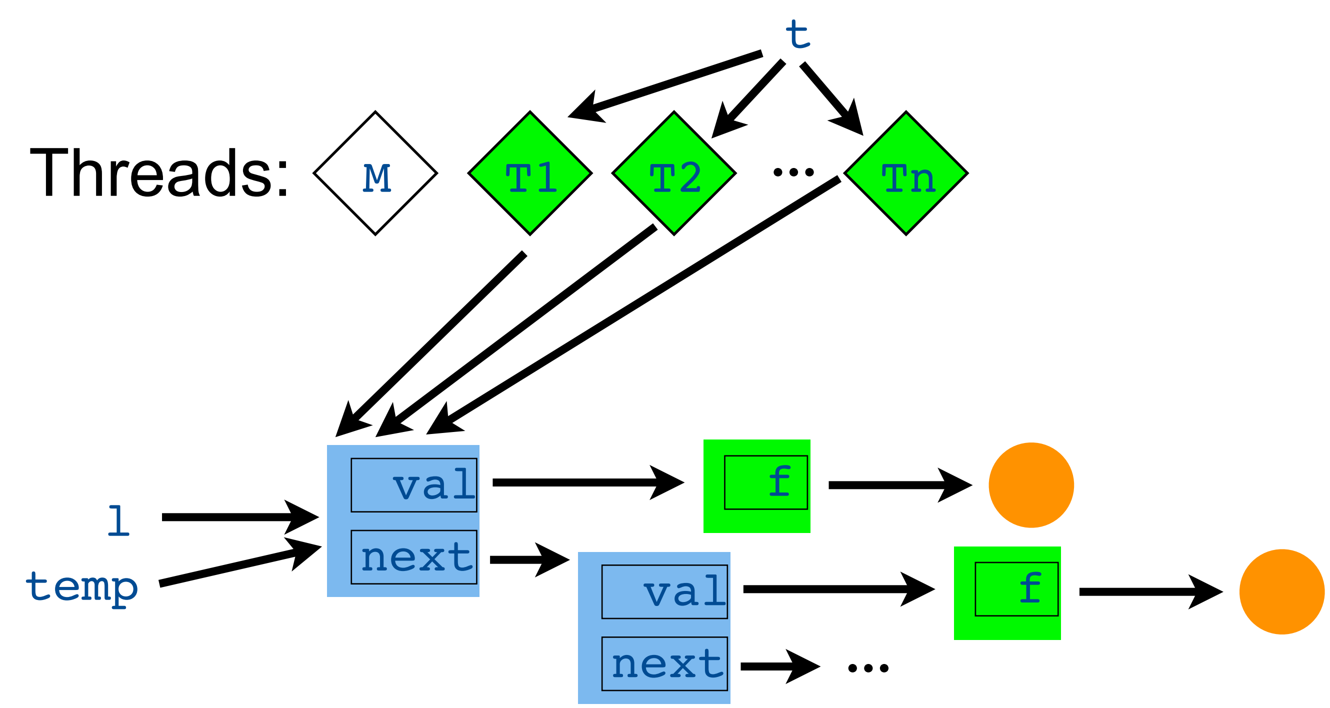
A Challenging Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`
2. We create several threads that all share the list `l`



Data Race Analysis

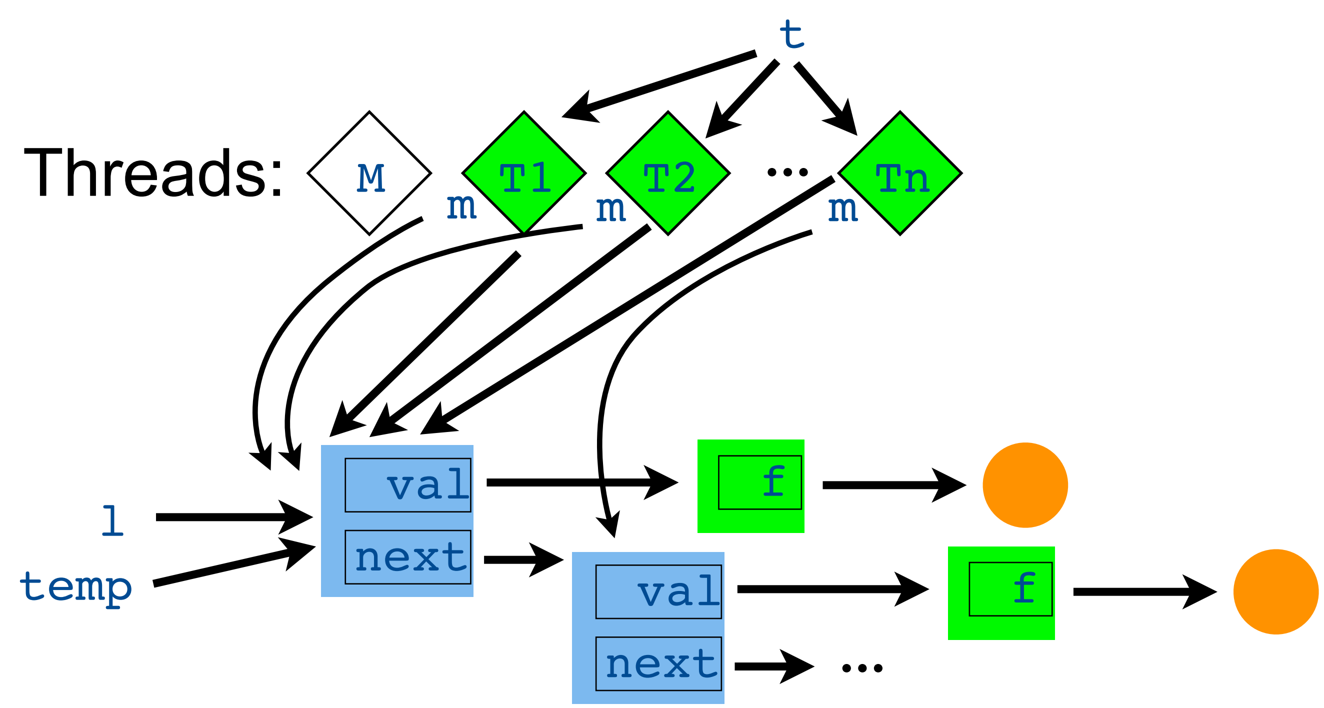
A Challenging Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`
2. We create several threads that all share the list `l`
3. Each thread chooses a cell, takes a lock on it and updates it.



Data Race Analysis

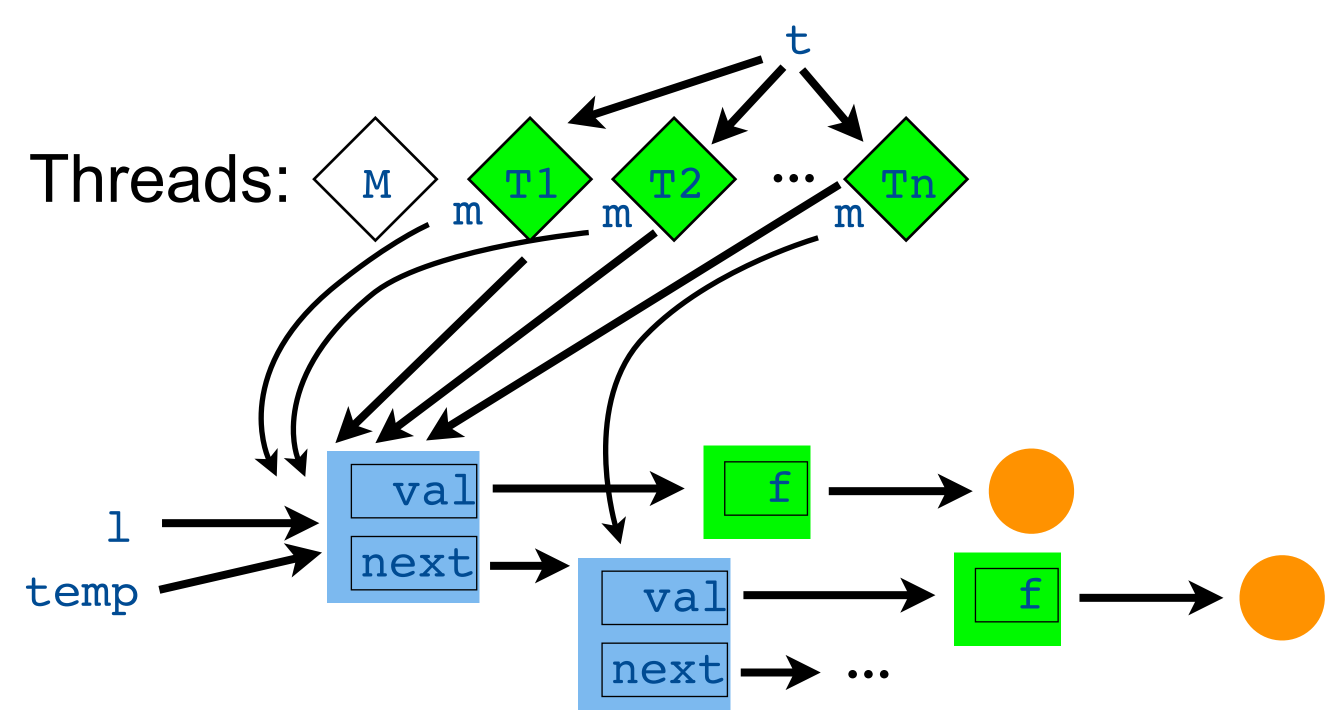
A Challenging Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

1. We create a link list `l`
2. We create several threads that all share the list `l`
3. Each thread chooses a cell, takes a lock on it and updates it.



Data Race Analysis

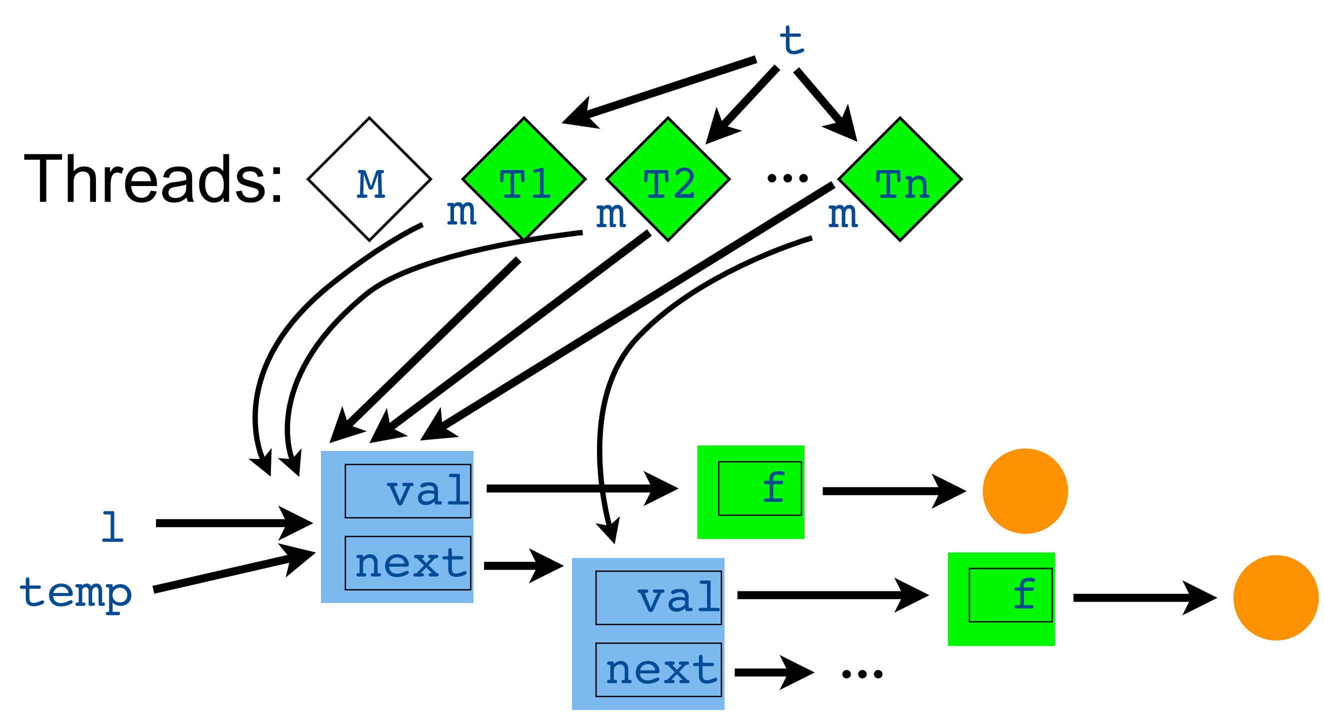
A Challenging Example

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Even on a simple program like this one, only a combination of static analyses will be able to prove data race freeness

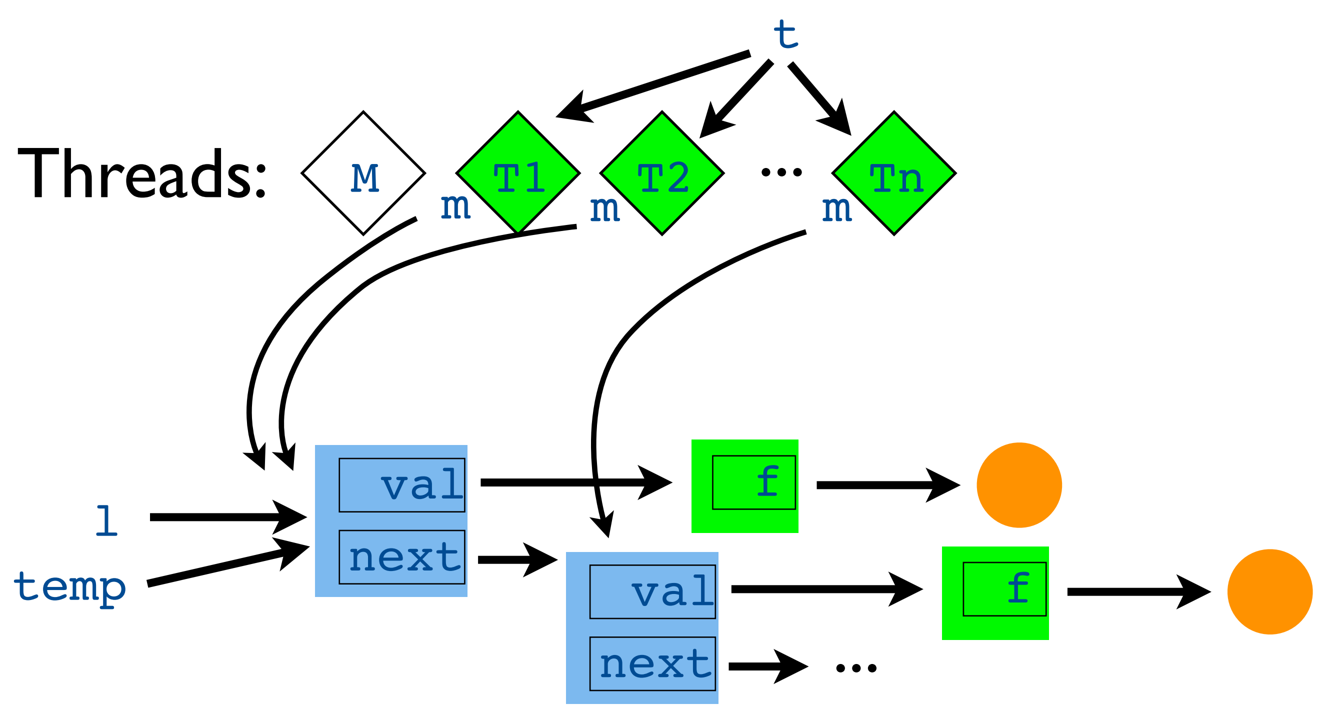


Points-to abstraction

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:     temp.next = l;
       l = temp }
    while (*) {
    h4 T t = new T();
4:     t.data = l;
       t.start();
5:     t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:     List m = this.data;
7:     while (*) { m = m.next; }
8:     synchronized(m){ m.val.f = ...;}}
    return;}}
```

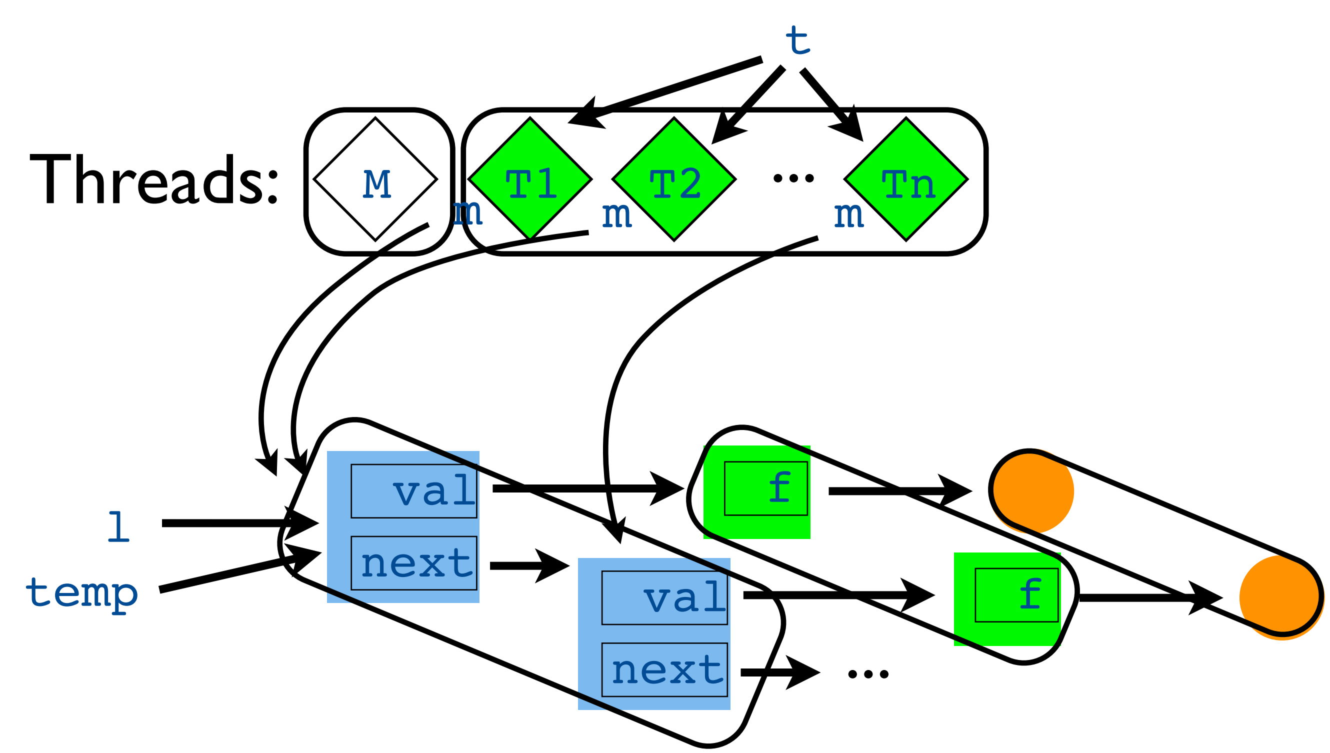


Points-to abstraction

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
    h1 List temp = new List();
1:  h2 temp.val = new T();
2:  h3 temp.val.f = new A();
3:     temp.next = l;
       l = temp }
    while (*) {
    h4 T t = new T();
4:     t.data = l;
       t.start();
5:     t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:     List m = this.data;
7:     while (*) { m = m.next; }
8:     synchronized(m){ m.val.f = ...;}}
    return;}}
```

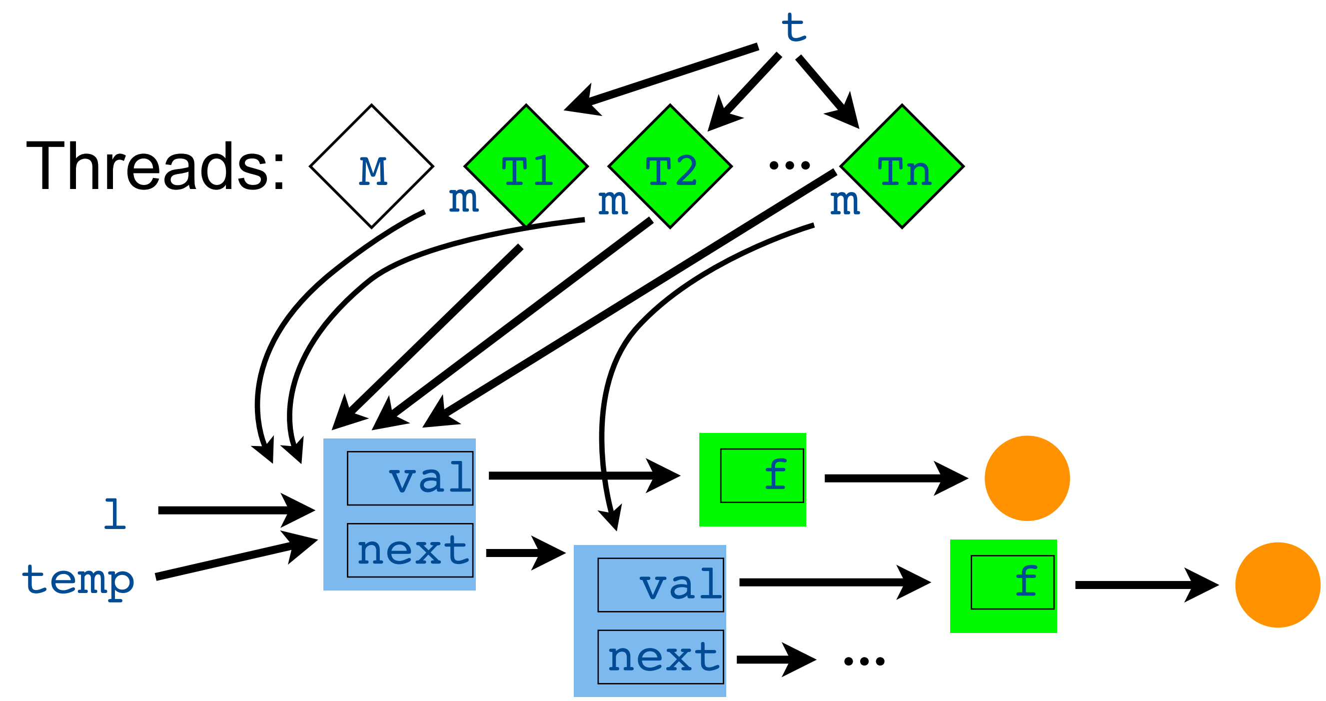


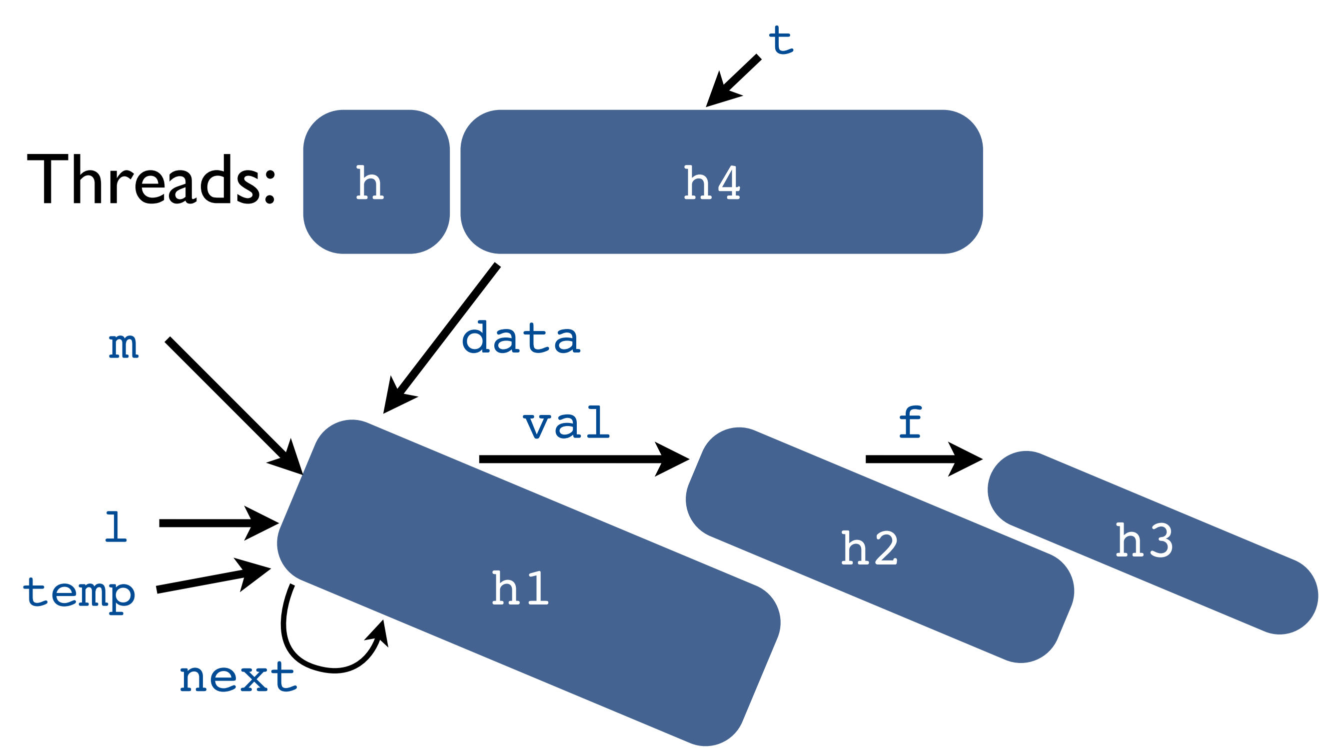
Points-to abstraction

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
h1  List temp = new List();
1:  h2  temp.val = new T();
2:  h3  temp.val.f = new A();
3:      temp.next = l;
        l = temp }
    while (*) {
h4  T t = new T();
4:      t.data = l;
        t.start();
5:      t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```





Effective Static Data Race Detection for Java

Naik's PhD 2008

- Original pairs
- Reachable pairs
- Aliasing pairs
- Escaping pairs
- Unlocked pairs

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

Reachable pairs

Aliasing pairs

Escaping pairs

Unlocked pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

The first set of potential races is based on field safety

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Reachable pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ ~~(5,f,5)~~ ~~(2,f,5)~~ (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

The pairs that may be reachable from the program entry and that may concern two distinct threads

Aliasing pairs

Escaping pairs

Unlocked pairs

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
  }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Reachable pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ ~~(5,f,5)~~ ~~(2,f,5)~~ (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Aliasing pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) ~~(2,f,5)~~ ~~(5,f,8)~~ (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Using a points-to abstraction we compute the pairs that may touch the same heap location

Escaping pairs

Unlocked pairs

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Reachable pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ ~~(5,f,5)~~ ~~(2,f,5)~~ (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Aliasing pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) ~~(2,f,5)~~ ~~(5,f,8)~~ (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Escaping pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ (5,f,5) ~~(2,f,5)~~ (5,f, 8) ~~(4,data,6)~~
~~(3,next, 7)~~ ~~(1,val, 8)~~ ~~(2,f,8)~~ (8,f, 8)

Unlocked pairs

The pairs of program points where the target location may be shared at that points

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Reachable pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ ~~(5,f,5)~~ ~~(2,f,5)~~ (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Aliasing pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) ~~(2,f,5)~~ ~~(5,f,8)~~ (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Escaping pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ (5,f,5) ~~(2,f,5)~~ (5,f, 8) ~~(4,data,6)~~
~~(3,next, 7)~~ ~~(1,val, 8)~~ ~~(2,f,8)~~ (8,f, 8)

Unlocked pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) ~~(8,f,8)~~

Pairs that may not be guarded by the same lock

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
      List temp = new List();
1:    temp.val = new T();
2:    temp.val.f = new A();
3:    temp.next = l;
      l = temp }
    while (*) {
      T t = new T();
4:    t.data = l;
      t.start();
5:    t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:    List m = this.data;
7:    while (*) { m = m.next; }
8:    synchronized(m){ m.val.f = ...;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Reachable pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ ~~(5,f,5)~~ ~~(2,f,5)~~ (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Aliasing pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) ~~(2,f,5)~~ ~~(5,f,8)~~ (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Escaping pairs

~~(1,val,1)~~ ~~(1,val,2)~~ ~~(2,f,2)~~ ~~(3,next,3)~~
~~(4,data,4)~~ (5,f,5) ~~(2,f,5)~~ (5,f, 8) ~~(4,data,6)~~
~~(3,next, 7)~~ ~~(1,val, 8)~~ ~~(2,f,8)~~ (8,f, 8)

Unlocked pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) ~~(8,f,8)~~

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
```

If each original pair has been removed at least one time, the program is data-race free

```
5:      t.f = ...;}
    return;
   }
}

class T extends java.lang.Thread {
  A f;
  List data;
  void run(){
    while(*){
6:      List m = this.data;
7:      while (*) { m = m.next; }
8:      synchronized(m){ m.val.f = ...;}}
    return;}}
```

Effective Static Data Race Detection for Java

Naik's PhD 2008

Original pairs

(1,~~val~~,1) (1,~~val~~,2) (2,~~f~~,2) (3,~~next~~,3)
(4,~~data~~,4) (5,~~f~~,5) (2,~~f~~,5) (5,~~f~~, 8) (4,~~data~~,6)
(3,~~next~~, 7) (1,~~val~~, 8) (2,~~f~~, 8) (8,~~f~~, 8)

Reachable pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Aliasing pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Escaping pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

Unlocked pairs

(1,val,1) (1,val,2) (2,f,2) (3,next,3)
(4,data,4) (5,f,5) (2,f,5) (5,f, 8) (4,data,6)
(3,next, 7) (1,val, 8) (2,f, 8) (8,f, 8)

```
class List{ T val; List next; }

class Main() {
  void main(){
    List l = null;
    while (*) {
```

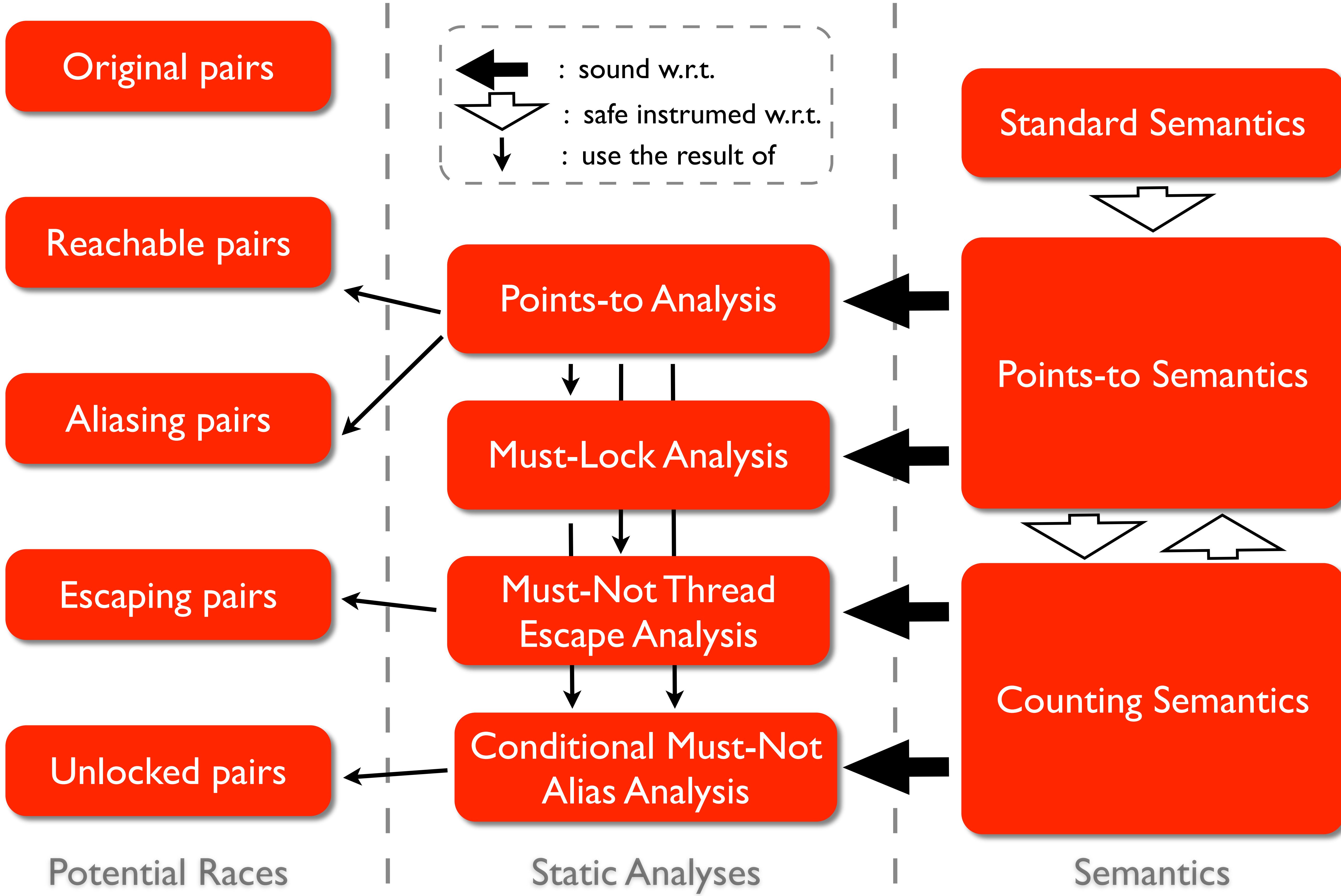
If each original pair has been removed at least one time, the program is data-race free

```
5:      t.f = ...;}
    return;
  }
```

This program is data-race free!

```
7:      while (*) { m = m.next; }
8:      synchronized(m){ m.val.f = ...;}}
    return;}}
```

Proof Architecture



Some reading

J. Choi, A. Loginov, and V. Sarkar. Static datarace analysis for multithreaded object-oriented programs. Tech. report, IBM Research Division, 2001.

M. Naik, A. Aiken, and J. Whaley. Effective static race detection for java. PLDI ’06

M. Naik and A. Aiken. Conditional must not aliasing for static race detection. POPL’07

M. Naik. Effective static race detection for java. PhD thesis, Stanford university, 2008.

C. Flanagan and S. N. Freund. *FastTrack: efficient and precise dynamic race detection.* Communication of the ACM 2010

H. Boehm and S. V. Adve. *You Don't Know Jack about Shared Variables or Memory Models.* Communication of the ACM 2012

Conclusions

Lessons learned

- Mechanized proof can handle more than toy static analyses
  - printing and proofreading these kind of proof would have been very difficult

- Realistic analyses are generally a composition of several sub-analyses
  - mechanized proof make explicit the interactions between them

- Proving correct of a realistic analysis is time consuming
  - about 1.5 man year effort and 15K loc for each proof...

Methodology

- Each verified analysis increases our knowledge about how to best formalize a static analysis in a proof assistant
  - avoid mechanically proving theorems that are not directly useful for soundness (termination, completeness)
  - design modular proofs with robust interfaces (intermediate semantics, module functors)

These Lectures

Lecture 1

Motivations

Examples of verified analysers

Lecture 2

Coq crash course

Lecture 3

Verified abstract interpreter for a simple imperative language

Lecture 4

CompCert

A verified value analysis for CompCert

A Posteriori Validation

[Extra slides if we have enough time]

A Posteriori Validation

An important tool in our toolbox

We program the *full* static analyzer inside Coq

```
Definition analyzer (p:program) :=
  ...
  let x := complex_computation p in
  ...
```

... or ...

A Posteriori Validation

An important tool in our toolbox

We program the *full* static analyzer inside Coq

```
Definition analyzer (p:program) :=
  ...
  let x := complex_computation p in
  ...
```

... or we program some parts in Coq, other parts in OCaml and use a verified validator

```
Definition analyzer (p:program) :=
  ...
  let x := ocaml_external_complex_computation p in
  match validator x p with
    | OK ⇒ ...
    | Error ⇒ abort
  end.
```

A Posteriori Validation

More formally, instead of proving a function $f \in A \rightarrow B$ satisfies a spec $R \subseteq A \times B$

$$\forall a \forall b, (a, f(a)) \in R$$

We define a validator $f? \in A \times B \rightarrow \{\text{true}, \text{false}\}$ and prove

$$\forall a \forall b, f?(a, b) = \text{true} \implies (a, b) \in R$$

Then every time we need a proof that $f(a)$ is correct for a given input a, we check if

$$f?(a, f(a)) = \text{true}$$

A Posteriori Validation

Example: CompCert register allocator

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=
  let g := interf_graph f live in
  let coloring := graph_coloring f g in
  if check_coloring g coloring
  then Some (alloc_of_coloring coloring g)
  else None.
```

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=

  let g := interf_graph f live in

  let coloring := graph_coloring f g in

  if check_coloring g coloring

  then Some (alloc_of_coloring coloring g)

  else None.
```

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

The function may fail!

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=

  let g := interf_graph f live in

  let coloring := graph_coloring f g in

  if check_coloring g coloring

  then Some (alloc_of_coloring coloring g)

  else None.
```

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

The function may fail!

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=
  let g := interf_graph f live in
  let coloring := graph_coloring f g in
  if check_coloring g coloring
  then Some (alloc_of_coloring coloring g)
  else None.
```

Computation of an interference graph between RTL registers: an edge between each registers with overlapping live ranges.

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

The function may fail!

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=
  let g := interf_graph f live in
  let coloring := graph_coloring f g in
  if check_coloring g coloring
  then Some (alloc_of_coloring coloring g)
  else None.
```

Computation of an interference graph between RTL registers: an edge between each registers with overlapping live ranges.

Graph coloring by an external program

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

The function may fail!

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=

  let g := interf_graph f live in

  let coloring := graph_coloring f g in

  if check_coloring g coloring

  then Some (alloc_of_coloring coloring g)

  else None.
```

Computation of an interference graph between RTL registers: an edge between each registers with overlapping live ranges.

Graph coloring by an external program

The validator simply verify that each edge connects nodes with different colors

A Posteriori Validation

Example: CompCert register allocator

The function takes a RTL function, the result of a live analysis and returns a location (machine register or in memory)

The function may fail!

```
Definition regalloc
  (f: function) (live: node → regset) : option (reg → loc) :=

  let g := interf_graph f live in

  let coloring := graph_coloring f g in

  if check_coloring g coloring

  then Some (alloc_of_coloring coloring g)

  else None.
```

Computation of an interference graph between RTL registers: an edge between each registers with overlapping live ranges.

Graph coloring by an external program

The validator simply verify that each edge connects nodes with different colors

If the validator fails, the whole function fails

A Posteriori Validation

More advanced pattern

More formally, instead of proving a function $f \in A \to B$ satisfies a spec $R \subseteq A \times B$

$$\forall a \forall b, (a, f(a)) \in R$$

We define a validator $f? \in A \times B \times C \to \{\text{true}, \text{false}\}$ and prove

$$\forall a \forall b \forall c, f?(a, b, c) = \text{true} \implies (a, b) \in R$$

Then every time we need a proof that $f(a)$ is correct for a given input a, we check if

$$f?(a, f(a), solver(a, f(a))) = \text{true}$$

A Posteriori Validation

More advanced pattern

More formally, instead of proving a function $f \in A \to B$ satisfies a spec $R \subseteq A \times B$

$$\forall a \forall b, (a, f(a)) \in R$$

We define a validator $f? \in A \times B \times C \to \{\text{true}, \text{false}\}$ and prove

$$\forall a \forall b \forall c, f?(a, b, c) = \text{true} \implies (a, b) \in R$$

computation hint

Then every time we need a proof that $f(a)$ is correct for a given input a, we check if

$$f?(a, f(a), solver(a, f(a))) = \text{true}$$

A Posteriori Validation

More advanced pattern

More formally, instead of proving a function $f \in A \to B$ satisfies a spec $R \subseteq A \times B$

$$\forall a \forall b, (a, f(a)) \in R$$

We define a validator $f? \in A \times B \times C \to \{\text{true}, \text{false}\}$ and prove

$$\forall a \forall b \forall c, f?(a, b, c) = \text{true} \implies (a, b) \in R$$

computation hint

Then every time we need a proof that $f(a)$ is correct for a given input a, we check if

$$f?(a, f(a), solver(a, f(a))) = \text{true}$$

untrusted

A Posteriori Validation

More advanced pattern

More formally, instead of proving a function $f \in A \to B$ satisfies a spec $R \subseteq A \times B$

$$\forall a \forall b, (a, f(a)) \in R$$

We define a validator $f? \in A \times B \times C \to \{\text{true}, \text{false}\}$ and prove

$$\forall a \forall b \forall c, f?(a, b, c) = \text{true} \implies (a, b) \in R$$

computation hint

Then every time we need a proof that $f(a)$ is correct for a given input a, we check if

$$f?(a, f(a), solver(a, f(a))) = \text{true}$$

untrusted

untrusted

A Posteriori Validation

More advanced example: Farkas proof

Some static analyses or optimisers need to prove unsat. of linear formula

$$\mathrm{UNSAT}(1 \leq x' + 2y \leq 4 \wedge x = x' + 1 \wedge x + 2y < 2)?$$

Just validating a yes/no result would be too hard

A Posteriori Validation

More advanced example: Farkas proof

Some static analyses or optimisers need to prove unsat. of linear formula

$$\mathrm{UNSAT}\left(\begin{array}{ccccc} -1 & +x' & & +2y & \geq & 0 \\ 0 & & & & \geq & 0 \\ -1 & -x' & +x & & = & 0 \\ 2 & & -x & -2y & > & 0 \end{array}\right)?$$

Just validating a yes/no result would be too hard

A Posteriori Validation

More advanced example: Farkas proof

Some static analyses or optimisers need to prove unsat. of linear formula

$$\text{UNSAT}\left(\begin{array}{ccccc} -1 & +x' & & +2y & \geq & 0 \\ 0 & & & & \geq & 0 \\ -1 & -x' & +x & & = & 0 \\ 2 & & -x & -2y & > & 0 \end{array}\right)?$$

Just validating a yes/no result would be too hard

$$\Uparrow$$

$$\text{UNSAT}\left(\begin{array}{cccccccc} 1 \times (& -1 & +x' & & +2y & \geq & 0) \\ 0 \times (& 4 & -x' & & -2y & \geq & 0) \\ -1 \times (& 1 & +x' & -x & & = & 0) \\ 1 \times (& 2 & & -x & -2y & > & 0) \end{array}\right)$$

Hint computed with an external tool (simplex)

A Posteriori Validation

More advanced example: Farkas proof

Some static analyses or optimisers need to prove unsat. of linear formula

$$\mathrm{UNSAT}\left(\begin{array}{rrrrcc} -1 & +x' & & +2y & \geq & 0 \\ 0 & & & & \geq & 0 \\ -1 & -x' & +x & & = & 0 \\ 2 & & -x & -2y & > & 0 \end{array}\right)?$$

Just validating a yes/no result would be too hard

$$\Uparrow$$

$$\mathrm{UNSAT}\left(\begin{array}{rrrrrcc} 1\times(& -1 & +x' & & +2y & \geq & 0) \\ 0\times(& 4 & -x' & & -2y & \geq & 0) \\ -1\times(& 1 & +x' & -x & & = & 0) \\ 1\times(& 2 & & -x & -2y & > & 0) \end{array}\right)$$

Hint computed with an external tool (simplex)

$$\Updownarrow$$

$$\mathrm{UNSAT}\left(\begin{array}{rrrrcc} -1 & +x' & & +2y & \geq & 0 \\ 4 & -x' & & -2y & \geq & 0 \\ 1 & +x' & -x & & = & 0 \\ 2 & & -x & -2y & > & 0 \end{array}\right)?$$

A Posteriori Validation

More advanced example: Farkas proof

Some static analyses or optimisers need to prove unsat. of linear formula

$$\text{UNSAT}\left(\begin{array}{ccccc} -1 & +x' & & +2y & \geq & 0 \\ 0 & & & & \geq & 0 \\ -1 & -x' & +x & & = & 0 \\ 2 & & -x & -2y & > & 0 \end{array}\right)?$$

Just validating a yes/no result would be too hard

$$\Uparrow$$

$$\text{UNSAT}\left(\begin{array}{ccccccc} 1 \times (& -1 & +x' & & +2y & \geq & 0) \\ 0 \times (& 4 & -x' & & -2y & \geq & 0) \\ -1 \times (& 1 & +x' & -x & & = & 0) \\ 1 \times (& 2 & & -x & -2y & > & 0) \end{array}\right)$$

Hint computed with an external tool (simplex)

$$\Updownarrow$$

$$\text{UNSAT}\,(0 > 0)?$$

A Posteriori Validation

Discussion

A validator may fail if

- The external tool contains bugs
  $\Rightarrow$ we must be able to abort the current computation
- The validator is not *smart* enough (incompleteness)

Some validator are *complete*

- The graph coloring validator is complete wrt any graph coloring algorithm
- The Farkas checker validator is complete if coefficients are rational (incomplete on integers)
- Other example: SSA generation
  In Gilles Barthe, Delphine Demange, and David Pichardie. *A formally verified SSA-based middle-end*, ESOP 2012, we provide a complete validator for SSA generation based on dominance frontier computation