Static analysis by abstract interpretation of run-time errors in synchronous and multithreaded embedded critical C programs

Antoine Miné

CNRS & École normale supérieure
Paris, France

MOVEP
CIRM, Marseille
4 December 2012

Motivation: Ariane 5, Flight 501

Maiden flight of the Ariane 5 Launcher, 4 June 1996.

Motivation: Ariane 5, Flight 501

40 s after launch...

Cause: uncaught exception after overflow in arithmetic overflow

Cost: 370 000 000 US$ [Dowson 97]

Review of verification methods

Review of verification methods

Testing

- well-established method
- but no formal warranty, high cost

Review of verification methods

Testing

- well-established method
- but no formal warranty, high cost

Formal methods:

Theorem proving

- proof essentially manual, but checked automatically
- powerful, but very steep learning curve and high human cost

Model checking

- checks a model of the program (usually user-specified, finite)
- automatic and complete (wrt. model), but often costly
- or automatic and incomplete (bounded model-checking)

Review of verification methods

Static analysis (by abstract interpretation)

- can work directly on the source code (not a model)
- automatic, always terminating, efficient
- parameterized by one/several abstraction(s)
- sound (full control and data coverage)
- incomplete (properties missed, false alarms)
- mostly used to check simple properties, with low precision requirements (e.g., for optimisation)

Review of verification methods

Static analysis (by abstract interpretation)

- can work directly on the source code (not a model)
- automatic, always terminating, efficient
- parameterized by one/several abstraction(s)
- sound (full control and data coverage)
- incomplete (properties missed, false alarms)
- mostly used to check simple properties, with low precision requirements (e.g., for optimisation)

Specialized static analyzer for validation

- checks for run-time errors (overflow, etc.)
- very precise on a chosen class of programs (no false alarm)
- gives sound results on all programs

Example static analyzers

Static analyzers checking for run-time errors in C code developed at ENS (Paris) in Patrick Cousot's group:

Astrée

- targets synchronous embedded real-time critical C code
- industrialized by AbsInt

AstréeA

- targets multithread embedded real-time C code
- research prototype in development

Related industrial tools elsewhere: PolySpace (MathWorks), cccheck (Microsoft), Sparrow, etc.

Outline

- analysis of non-parallel programs
  - abstract interpretation (in denotational form)
  - the Astrée analyzer
- analysis of multithreaded programs
  - abstracting interleavings with interferences (parallelism)
  - weak memory consistency (semantics of data-races)
  - thread synchronisation (mutexes and priorities)
  - the AstréeA prototype
  - limitations and possible extensions
- conclusion

Static analysis of non-parallel programs

Simple structured numeric language

Language syntax

$$
\begin{array}{lcll}
\textit{stat} & ::= & X \leftarrow \textit{expr} & \textit{(assignment)} \\
& | & \textbf{if } \textit{expr} \bowtie 0 \textbf{ then } \textit{stat} & \textit{(conditional)} \\
& | & \textbf{while } \textit{expr} \bowtie 0 \textbf{ do } \textit{stat} & \textit{(loop)} \\
& | & \textit{stat}; \textit{stat} & \textit{(sequence)} \\
\textit{expr} & ::= & X \mid [c_1, c_2] \mid \textit{expr} \diamond_\ell \textit{expr} \mid \cdots & \\
X \in \mathcal{V} & & \text{finite set of variables} & \\
\ell \in \mathcal{L} & & \text{syntactic locations} & \textit{(possible errors)}
\end{array}
$$

$c_1, c_2 \in \mathbb{R}$, $\diamond \in \{+, -, \times, /\}$, $\bowtie \in \{=, >, \geq, <, \leq\}$

Idealized language.

All variables are numeric and global. Functions are inlined.

Only possible error: division by zero.

Introduction to abstract interpretation

Abstract Interpretation

Abstract Interpretation

General theory of semantic approximation [Cousot Cousot 77,91]

Core principles:

- semantics are expressed as fixpoints (lfp F)
- semantics are linked through abstractions (α, γ)
- abstractions can be composed and reused (abstract domain)
- fixpoints can be approximated by iteration with acceleration (widening ∇)

Applications:

- compare existing semantics (unifying power)
- derive new semantics by abstraction
  derive computable semantics $\Longrightarrow$ **sound** static analysis

Static analysis by abstract interpretation

Road-map:

1. collecting concrete semantics lfp F in $\mathcal{D}$
   able to observe the properties of interest
   (hard to compute: large or infinite # of elements and chains in $\mathcal{D}$)
2. abstract domains
   - abstract values $\mathcal{D}^\sharp$: semantic choice + data-structures
     $\gamma : \mathcal{D}^\sharp \to \mathcal{D}$
   - abstract functions $F^\sharp$: algorithms + soundness proof
     $F(\gamma(X^\sharp)) \subseteq \gamma(F^\sharp(X^\sharp))$
   - convergence acceleration: ∇ + termination proof

Trace-based concrete semantics

Program states: $\sigma \in \Sigma \stackrel{\text{def}}{=} \mathcal{L} \times (\mathcal{E} \cup \{\omega\})$

- a control state in $\mathcal{L}$ (finite)
- a memory state in $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{V} \to \mathbb{R}$ (infinite), or an error ω
- initial states: $I \subseteq \Sigma$

Transition relation: $\to \in \Sigma \times \Sigma$

Trace semantics $\mathbb{T}$:

- set of execution traces, in $\mathcal{P}(\Sigma^*)$
- $\mathbb{T} \stackrel{\text{def}}{=}$ lfp F where
  $F(T) \stackrel{\text{def}}{=} I \cup \{ \langle \sigma_0, \ldots, \sigma_{n+1} \rangle \mid \langle \sigma_0, \ldots, \sigma_n \rangle \in T \wedge \sigma_n \to \sigma_{n+1} \}$

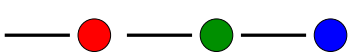
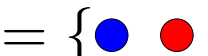
Computing $\mathbb{T}$ is generally undecidable.
(equivalent to exhaustive test)

State-based concrete semantics

State semantics $\mathbb{S}$:

- set of reachable states, in $\mathcal{P}(\Sigma)$
- $\mathbb{S} \overset{\text{def}}{=} \text{lfp}\, G$ where $G(S) \overset{\text{def}}{=} I \cup \{\, \sigma \mid \exists \sigma' \in S \wedge \sigma' \rightarrow \sigma \,\}$

The state-semantics is an abstraction of trace semantics

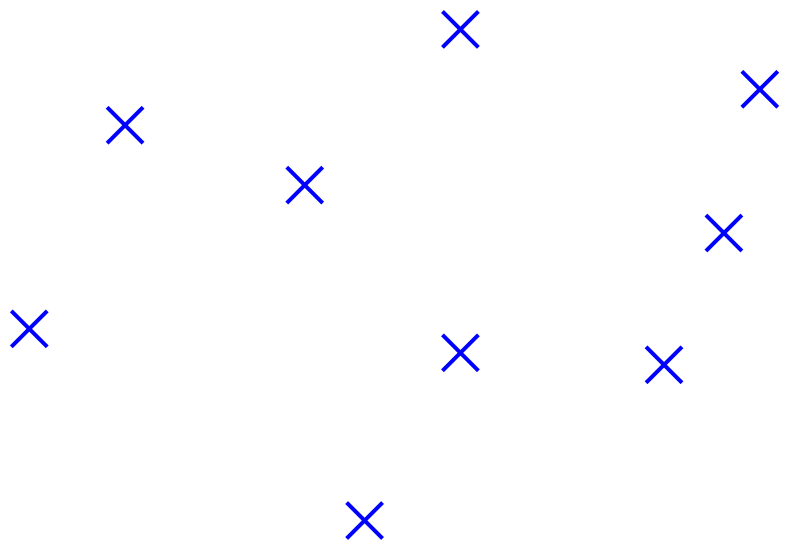
- we forget the ordering of states in traces
  $\alpha_{state}(T) \overset{\text{def}}{=} \{\, \sigma_i \mid \exists \langle \sigma_0, \ldots, \sigma_n \rangle \in T \wedge i \in [0, n] \,\}$
  e.g.: $\alpha_{state}(\{$  $\}) = \{$  $\}$
- $\mathbb{S} = \alpha_{state}(\mathbb{T})$
- $\gamma_{state} \circ G = F \circ \gamma_{state}$ where
  $\gamma_{state}(S) = \{\, \langle \sigma_0, \ldots, \sigma_n \rangle \mid \forall i \in [0, n],\, \sigma_i \in S \,\}$
- the abstraction is complete for safety properties

Computing $\mathbb{S}$ is undecidable or very costly.
(equivalent to exhaustive state-set exploration)

Numeric abstract domain

$\mathcal{P}(\Sigma) = \mathcal{P}(\mathcal{L} \times (\mathcal{E} \cup \{\omega\})) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\mathcal{L})$

$\Longrightarrow$ we abstract $\mathcal{P}(\mathcal{E}) \simeq \mathcal{P}(\mathbb{R}^{|\mathcal{V}|})$ further.

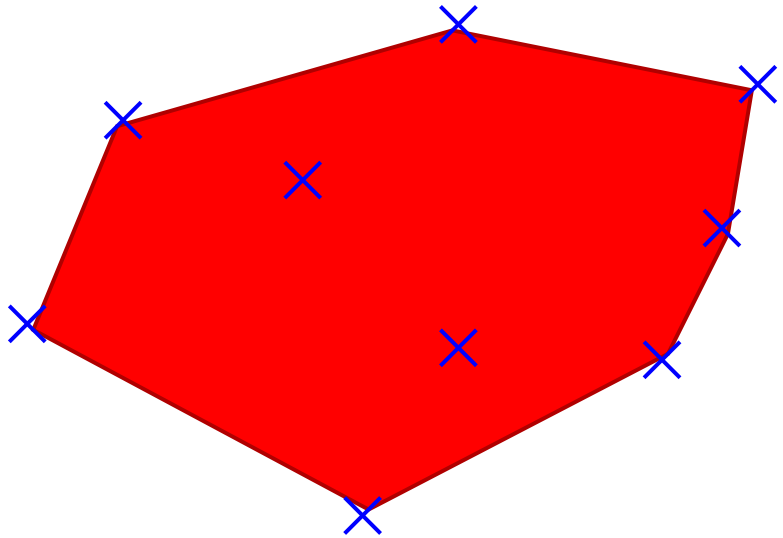


concrete sets $\mathcal{P}(\mathcal{E})$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

Numeric abstract domain

$$\mathcal{P}(\Sigma) = \mathcal{P}(\mathcal{L} \times (\mathcal{E} \cup \{\omega\})) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\mathcal{L})$$

$\Longrightarrow$ we abstract $\mathcal{P}(\mathcal{E}) \simeq \mathcal{P}(\mathbb{R}^{|\mathcal{V}|})$ further.



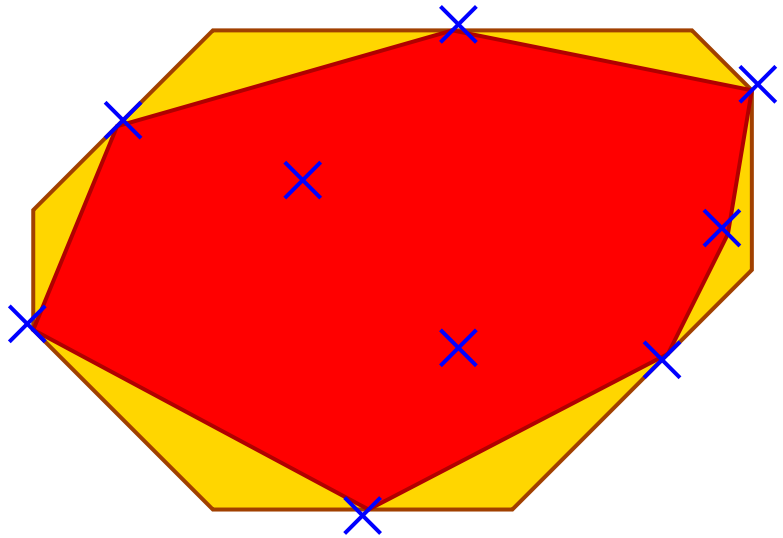
concrete sets $\mathcal{P}(\mathcal{E})$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

polyhedra $\mathcal{E}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

Numeric abstract domain

$$\mathcal{P}(\Sigma) = \mathcal{P}(\mathcal{L} \times (\mathcal{E} \cup \{\omega\})) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\mathcal{L})$$

$\Longrightarrow$ we abstract $\mathcal{P}(\mathcal{E}) \simeq \mathcal{P}(\mathbb{R}^{|\mathcal{V}|})$ further.



concrete sets $\mathcal{P}(\mathcal{E})$: $\{(0,3),(5.5,0),(12,7),\ldots\}$

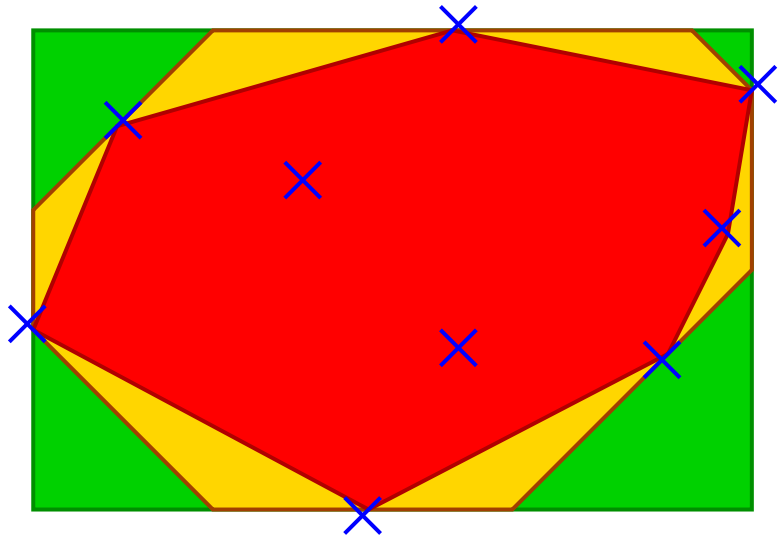
polyhedra $\mathcal{E}_p^\sharp$: $6X + 11Y \geq 33 \wedge \cdots$

octagons $\mathcal{E}_o^\sharp$: $X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$

Numeric abstract domain

$$\mathcal{P}(\Sigma) = \mathcal{P}(\mathcal{L} \times (\mathcal{E} \cup \{\omega\})) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\mathcal{L})$$

$\Longrightarrow$ we abstract $\mathcal{P}(\mathcal{E}) \simeq \mathcal{P}(\mathbb{R}^{|\mathcal{V}|})$ further.

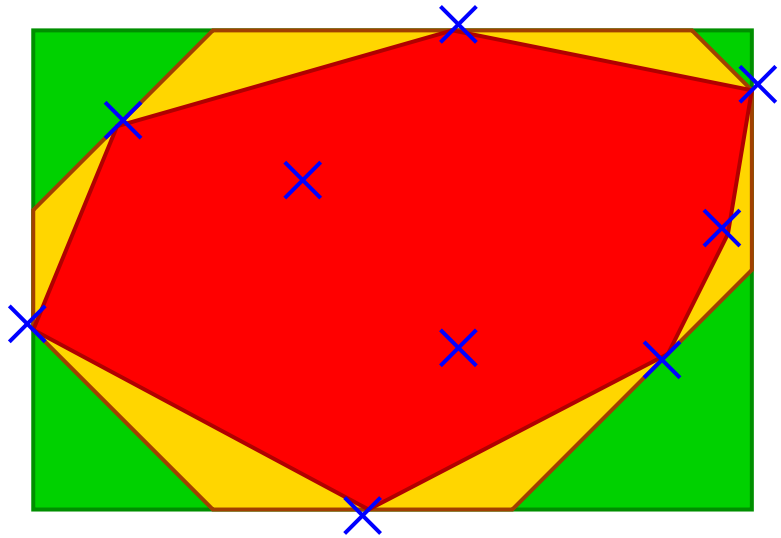


concrete sets $\mathcal{P}(\mathcal{E})$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$
polyhedra $\mathcal{E}_p^\sharp$:	$6X + 11Y \geq 33 \wedge \cdots$
octagons $\mathcal{E}_o^\sharp$:	$X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$
intervals $\mathcal{E}_i^\sharp$:	$X \in [0,12] \wedge Y \in [0,8]$

Numeric abstract domain

$$\mathcal{P}(\Sigma) = \mathcal{P}(\mathcal{L} \times (\mathcal{E} \cup \{\omega\})) \simeq (\mathcal{L} \to \mathcal{P}(\mathcal{E})) \times \mathcal{P}(\mathcal{L})$$

$\Longrightarrow$ we abstract $\mathcal{P}(\mathcal{E}) \simeq \mathcal{P}(\mathbb{R}^{|\mathcal{V}|})$ further.



concrete sets $\mathcal{P}(\mathcal{E})$:	$\{(0,3),(5.5,0),(12,7),\ldots\}$	not computable
polyhedra $\mathcal{E}_p^\sharp$:	$6X + 11Y \geq 33 \wedge \cdots$	exponential cost
octagons $\mathcal{E}_o^\sharp$:	$X + Y \geq 3 \wedge Y \geq 0 \wedge \cdots$	cubic cost
intervals $\mathcal{E}_i^\sharp$:	$X \in [0,12] \wedge Y \in [0,8]$	linear cost

Trade-off between cost and expressiveness / precision

Abstract interpretation in denotational form

State-based concrete semantics in denotational form

Expression semantics

$$\underline{\mathbb{E}[\![\, expr \,]\!] : \mathcal{E} \to (\mathcal{P}(\mathbb{R}) \times \mathcal{P}(\mathcal{L}))}$$

$$\mathbb{E}[\![\, X \,]\!]\rho \stackrel{\text{def}}{=} \langle \{\, \rho(X) \,\}, \emptyset \rangle$$

$$\mathbb{E}[\![\, [c_1, c_2] \,]\!]\rho \stackrel{\text{def}}{=} \langle \{\, x \in \mathbb{R} \,|\, c_1 \leq x \leq c_2 \,\}, \emptyset \rangle$$

$$\mathbb{E}[\![\, e_1 /_\ell\, e_2 \,]\!]\rho \stackrel{\text{def}}{=}$$

$$\text{let } \forall i \in \{1,2\}, \langle V_i, \Omega_i \rangle = \mathbb{E}[\![\, e_i \,]\!]\rho \text{ in}$$

$$\langle \{\, v_1/v_2 \,|\, v_i \in V_i,\ v_2 \neq 0 \,\},\ \Omega_1 \cup \Omega_2 \cup \{\, \ell \text{ if } 0 \in V_2 \,\} \rangle$$

$\ldots$

- input: memory states $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{V} \to \mathbb{R}$
- output:
  - set of values $V \subseteq \mathbb{R}$ (non-determinism)
  - set of error locations $\Omega \subseteq \mathcal{L}$

State-based concrete semantics in denotational form

Statement semantics

$$\mathbb{S}[\![\, stat \,]\!] : \mathcal{D} \xrightarrow{\sqcup} \mathcal{D}$$

$$(\text{noting } \langle\, V_\rho, \Omega_\rho \,\rangle \overset{\text{def}}{=} \mathbb{E}[\![\, e \,]\!]\rho)$$

$$\mathbb{S}[\![\, X \leftarrow e \,]\!]\langle\, R, \Omega \,\rangle \overset{\text{def}}{=} \langle\, \emptyset, \Omega \,\rangle \sqcup \bigsqcup_{\rho \in R} \langle\, \{\, \rho[X \mapsto v] \mid v \in V_\rho \,\}, \Omega_\rho \,\rangle$$

$$\mathbb{S}[\![\, e \bowtie 0? \,]\!]\langle\, R, \Omega \,\rangle \overset{\text{def}}{=} \langle\, \emptyset, \Omega \,\rangle \sqcup \bigsqcup_{\rho \in R} \langle\, \{\, \rho \mid \exists v \in V_\rho,\ v \bowtie 0 \,\}, \Omega_\rho \,\rangle$$

$$\mathbb{S}[\![\, \textbf{if } e \bowtie 0 \textbf{ then } s \,]\!]X \overset{\text{def}}{=} (\mathbb{S}[\![\, s \,]\!] \circ \mathbb{S}[\![\, e \bowtie 0? \,]\!])X \sqcup \mathbb{S}[\![\, e \not\bowtie 0? \,]\!]X$$

$$\mathbb{S}[\![\, \textbf{while } e \bowtie 0 \textbf{ do } s \,]\!]X \overset{\text{def}}{=} \mathbb{S}[\![\, e \not\bowtie 0? \,]\!](\text{lfp}\lambda Y.\, X \sqcup (\mathbb{S}[\![\, s \,]\!] \circ \mathbb{S}[\![\, e \bowtie 0? \,]\!])Y)$$

$$\mathbb{S}[\![\, s_1 ; s_2 \,]\!] \overset{\text{def}}{=} \mathbb{S}[\![\, s_2 \,]\!] \circ \mathbb{S}[\![\, s_1 \,]\!]$$

- $\mathcal{D} \overset{\text{def}}{=} \mathcal{P}(\mathcal{E}) \times \mathcal{P}(\mathcal{L})$, with pointwise join $\sqcup$, order $\sqsubseteq$
- $\mathbb{S}[\![\, stat \,]\!]$ defined by structural induction on the syntax
  - mutate memory states $\rho \in R \subseteq \mathcal{E}$
  - accumulate error locations $\Omega \subseteq \mathcal{L}$
  - complete $\sqcup$−morphism

State-based concrete semantics in denotational form

Program semantics

$$\mathbb{P} \stackrel{\text{def}}{=} \left[\mathbb{S}[\![\mathit{prog}]\!] \langle \mathcal{E}_0, \emptyset \rangle \right]_\Omega$$

- $\mathit{prog} \in \mathit{stat}$
- start form initial memory states $\mathcal{E}_0 \subseteq \mathcal{E}$
- output only error locations, in $\mathcal{L}$

Static analysis as abstract interpretation

Abstract semantics of statements

$\mathbb{S}^\sharp[\![stat]\!] : \mathcal{D}^\sharp \to \mathcal{D}^\sharp$

$\mathbb{S}^\sharp[\![X \leftarrow e]\!], \mathbb{S}^\sharp[\![e \bowtie 0?]\!]$ are given

$\mathbb{S}^\sharp[\![\textbf{if } e \bowtie 0 \textbf{ then } s]\!] X^\sharp \stackrel{\text{def}}{=} (\mathbb{S}^\sharp[\![s]\!] \circ \mathbb{S}^\sharp[\![e \bowtie 0?]\!]) X^\sharp \sqcup^\sharp \mathbb{S}^\sharp[\![e \not\bowtie 0?]\!] X^\sharp$

$\mathbb{S}^\sharp[\![\textbf{while } e \bowtie 0 \textbf{ do } s]\!] X^\sharp \stackrel{\text{def}}{=}$
$\mathbb{S}^\sharp[\![e \not\bowtie 0?]\!](\text{lim } \lambda Y^\sharp.\ Y^\sharp \triangledown (X^\sharp \sqcup^\sharp (\mathbb{S}^\sharp[\![s]\!] \circ \mathbb{S}^\sharp[\![e \bowtie 0?]\!]) Y^\sharp))$

$\mathbb{S}^\sharp[\![s_1; s_2]\!] \stackrel{\text{def}}{=} \mathbb{S}^\sharp[\![s_2]\!] \circ \mathbb{S}^\sharp[\![s_1]\!]$

- $\mathcal{D}^\sharp \stackrel{\text{def}}{=} \mathcal{E}^\sharp \times \mathcal{P}(\mathcal{L})$, where $\mathcal{E}^\sharp$ is a numeric abstract domain.
- $\gamma_{\mathcal{D}}\langle E^\sharp, \Omega \rangle \stackrel{\text{def}}{=} \langle \gamma_{\mathcal{E}}(E^\sharp), \Omega \rangle$
- $\mathbb{S}^\sharp[\![s]\!]$, $\sqcup^\sharp$ sound abstractions of $\mathbb{S}[\![s]\!]$, $\sqcup$:
$$\mathbb{S}[\![s]\!] \circ \gamma \sqsubseteq \gamma \circ \mathbb{S}^\sharp[\![s]\!]$$
- approximate fixpoint solutions by iteration with widening $\triangledown$

Static analysis as abstract interpretation

Abstract semantics of programs

$$\mathbb{P}^\sharp \stackrel{\text{def}}{=} \left[\mathbb{S}^\sharp[\![\mathit{prog}]\!] \langle \mathcal{E}^\sharp_0, \emptyset \rangle \right]_\Omega$$

The analysis is:

- effective
- defined by structural induction on the program syntax
- efficient in memory (parsimonious use of abstract states)
- fully flow-sensitive
- parametrized by the choice of an abstract domain $\mathcal{E}^\sharp$
- sound: $\mathbb{P} \subseteq \mathbb{P}^\sharp$

Example domain: the interval domain

$$\mathcal{E}^\sharp \stackrel{\text{def}}{=} \mathcal{V} \to (\mathbb{R} \cup \{-\infty\}) \times (\mathbb{R} \cup \{+\infty\})$$

- $X^\sharp \in \mathcal{E}^\sharp$ maps each variable to a pair of bounds
  $\gamma_{\mathcal{E}}(X^\sharp) \stackrel{\text{def}}{=} \{ \rho \in \mathcal{E} \mid \forall V \in \mathcal{V},\ \mathit{fst}(X^\sharp(V)) \leq \rho(V) \leq \mathit{snd}(X^\sharp(V)) \}$
  (can express absence of arithmetic and array overflow, etc.)
- $\mathbb{S}^\sharp[\![X \leftarrow e]\!]$: interval arithmetics
- $\mathbb{S}^\sharp[\![e \bowtie 0?]\!]$: box tightening
- $\sqcup^\sharp$: box hull (keep loosest bounds)
- $\triangledown$: set unstable bounds to $\pm\infty$ (or use thresholds)
  e.g.: $[0,1] \triangledown [0,2] = [0,+\infty]$
- easy to adapt to machine integer and floating-point semantics

The need for relational domains

Example

$i \leftarrow 0;\ x \leftarrow 0;$
while $i < 1000000$ **do**
    if $[0,1] = 0$ **then** $x \leftarrow x + [-1,1];$
    $i \leftarrow i + 1$

Causes:

- approximations accumulate along abstract executions
  (the combination of optimal abstract operators is not optimal)
- extrapolations ∇ introduce extra approximations
- the need to find inductive loop invariants of a complex form
  $(-i \leq x \leq i)$

An (infinite) abstract domain works on infinitely many programs and fails on infinitely many programs!

Astrée

Specialized static analyzers

Design by refinement:

- focus on a specific family of programs and properties
- start with a fast and coarse analyzer (intervals)
- while the precision is insufficient (too many false alarms)
  - add new abstract domains (generic or application-specific)
  - refine existing domains (better transfer functions)
  - improve communication between domains (reductions)

$\Longrightarrow$ analyzer specialized for a (infinite) class of programs

- efficient and precise
- parametric (by end-users, to analyze new programs in the family)
- extensible (by developers, to analyze related families)

The Astrée static analyzer

Analyseur statique de programmes temps-réels embarqués
(static analyzer for real-time embedded software)

- developed at ENS (since 2001)
  > B. Blanchet, P. Cousot, R. Cousot, J. Feret,
  > L. Mauborgne, D. Monniaux, A. Miné, X. Rival
- industrialized and made commercially available by AbsInt (since 2009)

Astrée

`www.astree.ens.fr`

AbsInt

`www.absint.com`

Astrée's concrete semantics

Concrete semantics source

- C99 norm (portable programs)
- IEEE 754-1985 norm (floating-point arithmetic)
- architecture parameters (`sizeof`, endianess, `struct`, etc.)
- compiler and linker parameters (initialization, etc.)

Limitations

- no dynamic memory allocation
- no recursivity
- stand-alone programs only (no external libraries)
- monolithic analysis (non-modular)
- no parallel program

Astrée's concrete semantics

Run-time errors

- overflows in float, integer, `enum` arithmetic and cast
- division, modulo by 0 on integers and floats
- invalid argument of bit-shift
- out-of-bound array access
- invalid pointer arithmetic or dereferencing
- violation of user-specified assertions (`assert`)

Semantics after an error:

- halt the program (out of bound array access)
- return a specific value (modulo after arithmetic overflow)
- return all values in the type (invalid bit-shift)

$\Longrightarrow$ try to continue the analysis after an alarm

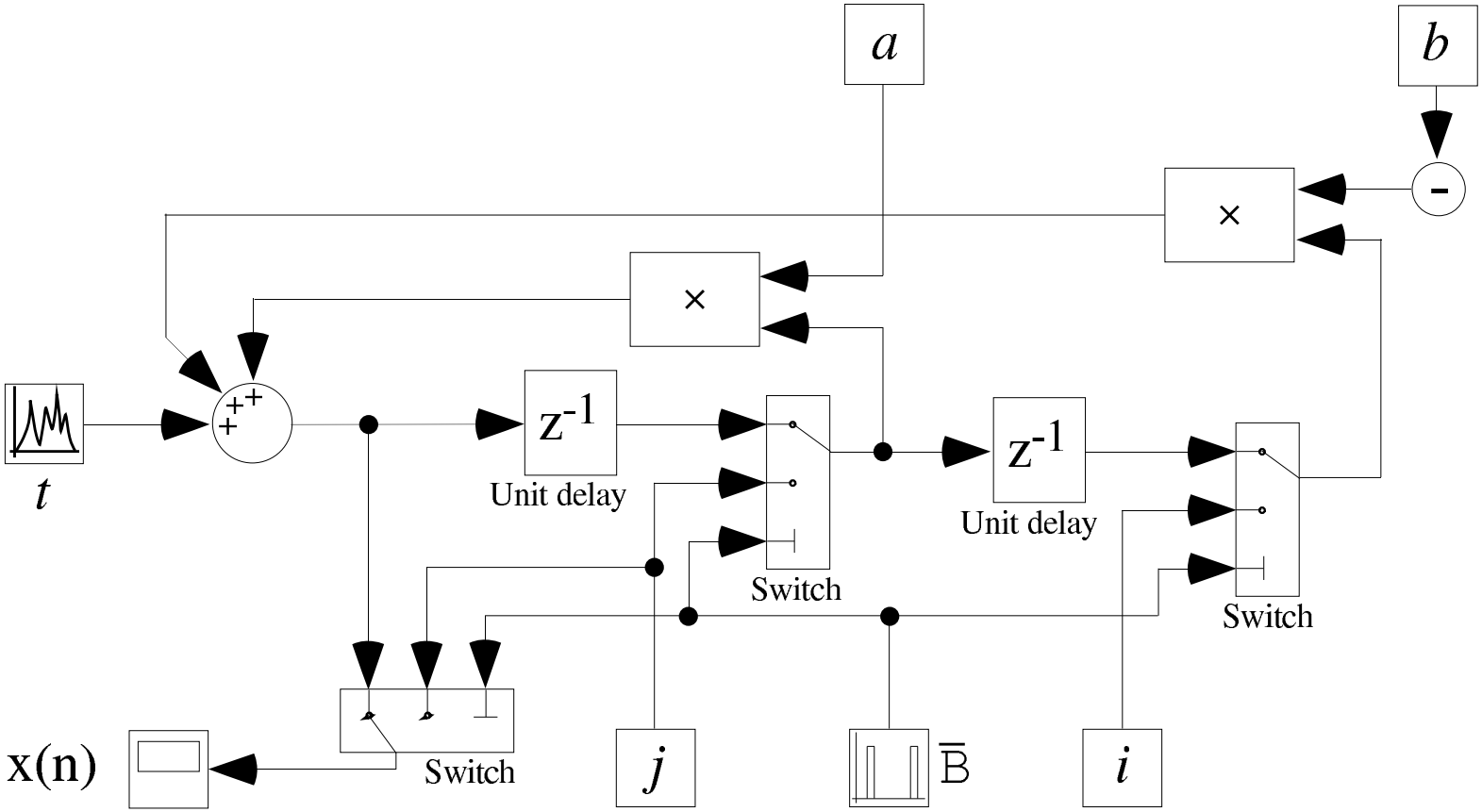
Target software

- synchronous reactive codes
- avionics control/command codes
- compiled to C from a graphical language *a la Scade / Simulink*

Structure

initialize state variables
`while ( clock` $\leq$ `3600000 )` {
    read input from sensors (`volatile`)
    compute output and new state
    write output to actuators
    wait for clock tick
}

Target software



Second order digital filter in Simulink

Abstract interpreter

syntax iterator

↕

trace partitioning domain

↕

pointer domain

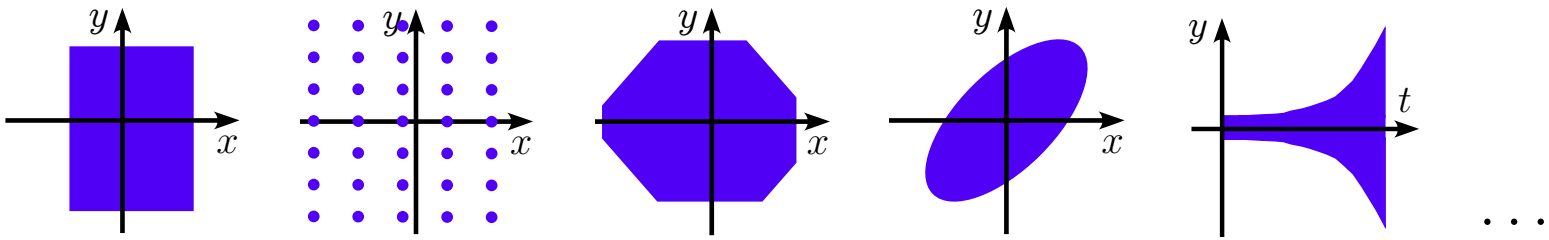
↕

(reduced product of) numerical abstract domains

↕ intervals ↕ octagons ↕ decision trees ↕ filters ⋮ ...

↕

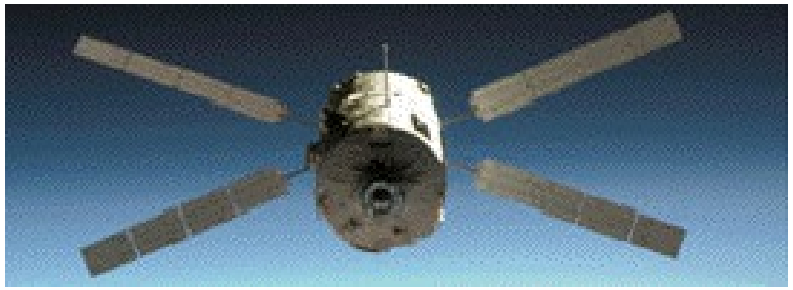
intervals



Astrée applications

Airbus A340-300 (2003)

Airbus A380 (2004)



(model of) ESA ATV (2008)

- size: from 70 000 to 860 000 lines of C
- analysis time: from 45mn to $\simeq$40h
- alarm(s): 0 (proof of absence of run-time error)

Static analysis of multithreaded programs

Embedded multithreaded software

Target: embedded multithreaded programs

- fixed set of threads (no dynamic creation of thread)
- communicating in shared memory
- scheduled on a single, mono-core processor
- running a real-time OS

Why?

- trend to use multithreaded code even in critical applications (e.g., Integrated Modular Avionics)
- testing is not effective on multithreaded programs
  - interleaving $\Rightarrow$ combinatorial blow-up of executions
    $\Rightarrow$ tests have very low coverage
  - bugs appear in corner cases (data-races)

Control path based semantics

Control paths

$atomic ::= X \leftarrow expr \mid expr \bowtie 0?$

Control paths

$\pi : stat \to \mathcal{P}(atomic^*)$

$\pi(X \leftarrow e) \stackrel{\text{def}}{=} \{ X \leftarrow e \}$

$\pi(\textbf{if } e \bowtie 0 \textbf{ then } s) \stackrel{\text{def}}{=} (\{ e \bowtie 0? \} \cdot \pi(s)) \cup \{ e \not\bowtie 0? \}$

$\pi(\textbf{while } e \bowtie 0 \textbf{ do } s) \stackrel{\text{def}}{=} \left(\bigcup_{i \geq 0} (\{ e \bowtie 0? \} \cdot \pi(s))^i \right) \cdot \{ e \not\bowtie 0 \}?$

$\pi(s_1; s_2) \stackrel{\text{def}}{=} \pi(s_1) \cdot \pi(s_2)$

$\pi(prog)$ is a (generally infinite) set of finite control paths

Path-based concrete semantics of sequential programs

Join-over-all-path semantics

$$\underline{\Pi[\![P]\!] : \mathcal{D} \overset{\sqcup}{\to} \mathcal{D}} \quad P \subseteq \mathit{atomic}^*$$

$$\Pi[\![P]\!]\langle R, \Omega \rangle \overset{\text{def}}{=} \bigsqcup_{s_1 \cdot \ldots \cdot s_n \in P} (\mathbb{S}[\![s_n]\!] \circ \cdots \circ \mathbb{S}[\![s_1]\!])\langle R, \Omega \rangle$$

Semantic equivalence

$$\mathbb{S}[\![\mathit{prog}]\!] = \Pi[\![\pi(\mathit{prog})]\!]$$

(not true in the abstract)

Advantages:

- easily extended to parallel programs (path interleavings)
- allows reasoning about local program transformations (weak memory models)

Path-based concrete semantics of parallel programs

Finite, fixed set of threads: $\mathcal{T} \stackrel{\text{def}}{=} \{ t_1, \ldots, t_n \}$, $prog_{t_i} \in stat$

Parallel control paths

$$\begin{aligned}\pi_* &\stackrel{\text{def}}{=} \{ \text{interleavings of } \pi(prog_t),\ t \in \mathcal{T} \} \\ &= \{ p \in atomic^* \mid \forall t \in \mathcal{T},\ proj_t(p) \in \pi(prog_t) \}\end{aligned}$$

Interleaving program semantics

$$\mathbb{P}_* \stackrel{\text{def}}{=} [\, \sqcap [\![\pi_*]\!] \langle \mathcal{E}_0, \emptyset \rangle \,]_\Omega$$

($\simeq$ sequentially consistent executions [Lamport 79])

Issues:

- too many paths to consider exhaustively
- no induction structure to iterate on
- unrealistic assumptions on granularity and memory consistency

Example

Example

$\mathcal{E}_0 : x = y = 0$

f :	①	**while** $0 = 0$ **do**	g :	①	**while** $0 = 0$ **do**
	②	**if** $x < y$ **then**		②	**if** $y < 10$ **then**
	③	$x \leftarrow x + 1$		③	$y \leftarrow y + 1$

At ②, we have $0 \leq x \leq y \leq 10$.

Abstracting interleavings as interferences

Detour to rely/guarantee proof methods

checking f

① **while** *true* **do**	x unchanged
② **if** $x < y$ **then**	y incremented
③ $x \leftarrow x + 1$	$y \leq 10$

checking g

x unchanged	① **while** *true* **do**
y unchanged	② **if** $y < 10$ **then**
	③ $y \leftarrow y + 1$

invariant at ②: $0 \leq x \leq y \leq 10$

Modular proof method [Jones 81]

Annotate programs with:

- local invariants
- rely and guarantee on transitions

For each thread, prove that:

- local and guarantee hold
- using rely

rely/guarantee act as thread interfaces (abstraction)

How do we infer instead of relying on annotations?

Concrete denotational semantics

Interferences in $\mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathcal{V} \times \mathbb{R}$

$\langle t, X, v \rangle$ means: t can store the value v into the variable X

Interference program semantics

$$\mathbb{E}_{\mathcal{I}}[\![X]\!]_t\langle \rho, I \rangle \stackrel{\text{def}}{=} \{ \rho(X) \} \cup \{ v \mid \exists t' \neq t, \langle t', X, v \rangle \in I \}$$

$$\mathbb{S}_{\mathcal{I}}[\![X \leftarrow e]\!]_t\langle R, \Omega, I \rangle \stackrel{\text{def}}{=}$$
$$\langle \emptyset, \Omega, I \rangle \sqcup \bigsqcup_{\rho \in R} \langle \{ \rho[X \mapsto v] \mid v \in V_\rho \}, \Omega_\rho, \{ \langle t, X, v \rangle \mid v \in V_\rho \} \rangle$$
where $\langle V_\rho, \Omega_\rho \rangle = \mathbb{E}_{\mathcal{I}}[\![e]\!]_t\langle \rho, I \rangle$

$$\mathbb{P}_{\mathcal{I}} \stackrel{\text{def}}{=} \left[\text{lfp } \lambda\langle \Omega, I \rangle . \bigsqcup_{t \in \mathcal{T}} [\mathbb{S}_{\mathcal{I}}[\![prog_t]\!]_t\langle \mathcal{E}_0, \emptyset, I \rangle]_{\Omega, I} \right]_\Omega$$

- $\mathbb{E}_{\mathcal{I}}$, $\mathbb{S}_{\mathcal{I}}$ use and enrich $I \subseteq \mathcal{I}$
- threads are re-analyzed until interferences I stabilize (lfp)

Example

Example

$\mathcal{E}_0 : x = y = 0$

f :	①	**while** $0 = 0$ **do**	g :	①	**while** $0 = 0$ **do**
	②	**if** $x < y$ **then**		②	**if** $y < 10$ **then**
	③	$x \leftarrow x + 1$		③	$y \leftarrow y + 1$

Concrete interference semantics:

iteration 1

$I = \emptyset$

② : $x = 0,\ y = 0$

② : $x = 0,\ y \in [0, 10]$

new $I = \{ \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

Example

Example

$\mathcal{E}_0 : x = y = 0$

f :	①	**while** $0 = 0$ **do**	g :	①	**while** $0 = 0$ **do**
	②	**if** $x < y$ **then**		②	**if** $y < 10$ **then**
	③	$x \leftarrow x + 1$		③	$y \leftarrow y + 1$

Concrete interference semantics:

iteration 2

$I = \{ \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

② : $x \in [0, 10]$, $y = 0$

② : $x = 0$, $y \in [0, 10]$

new $I = \{ \langle f, x, 1 \rangle, \ldots, \langle f, x, 10 \rangle, \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

Example

Example

$\mathcal{E}_0 : x = y = 0$

f :	①	**while** $0 = 0$ **do**	g :	①	**while** $0 = 0$ **do**
	②	**if** $x < y$ **then**		②	**if** $y < 10$ **then**
	③	$x \leftarrow x + 1$		③	$y \leftarrow y + 1$

Concrete interference semantics:

iteration 3

$I = \{ \langle f, x, 1 \rangle, \ldots, \langle f, x, 10 \rangle, \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

② : $x \in [0, 10]$, $y = 0$

② : $x = 0$, $y \in [0, 10]$

new $I = \{ \langle f, x, 1 \rangle, \ldots, \langle f, x, 10 \rangle, \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

Example

Example

$\mathcal{E}_0 : x = y = 0$

f :	①	**while** $0 = 0$ **do**	g :	①	**while** $0 = 0$ **do**
	②	**if** $x < y$ **then**		②	**if** $y < 10$ **then**
	③	$x \leftarrow x + 1$		③	$y \leftarrow y + 1$

Concrete interference semantics:

iteration 3

$I = \{ \langle f, x, 1 \rangle, \ldots, \langle f, x, 10 \rangle, \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

② : $x \in [0, 10]$, $y = 0$

② : $x = 0$, $y \in [0, 10]$

new $I = \{ \langle f, x, 1 \rangle, \ldots, \langle f, x, 10 \rangle, \langle g, y, 1 \rangle, \ldots, \langle g, y, 10 \rangle \}$

Note: we don't get that $x \leq y$ at ②, only that $x, y \in [0, 10]$

Soundness of the interference semantics

Soundness theorem

$$\mathbb{P}_* \subseteq \mathbb{P}_{\mathcal{I}}$$

Proof sketch:

- define $\mathbb{\Pi}_{\mathcal{I}}[\![P]\!]_t A \stackrel{\text{def}}{=} \bigsqcup \{ \mathbb{S}_{\mathcal{I}}[\![s_1; \ldots; s_n]\!] A \mid s_1 \cdot \ldots \cdot s_n \in P \}$; then $\mathbb{\Pi}_{\mathcal{I}}[\![\pi(s)]\!]_t = \mathbb{S}_{\mathcal{I}}[\![s]\!]_t$;
- given the interference fixpoint $I \subseteq \mathcal{I}$ from $\mathbb{P}_{\mathcal{I}}$, prove by recurrence on the length of $p \in \pi_*$ that:
  - $\forall t \in \mathcal{T}, \forall \rho \in [\mathbb{\Pi}[\![p]\!] \langle \mathcal{E}_0, \emptyset \rangle]_{\mathcal{E}}$, $\exists \rho' \in [\mathbb{\Pi}_{\mathcal{I}}[\![proj_t(p)]\!]_t \langle \mathcal{E}_0, \emptyset, I \rangle]_{\mathcal{E}}$ such that $\forall X \in \mathcal{V}$, $\rho(X) = \rho'(X)$ or $\langle t', X, \rho(X) \rangle \in I$ for some $t' \neq t$.
  - $[\mathbb{\Pi}[\![p]\!] \langle \mathcal{E}_0, \emptyset \rangle]_\Omega \subseteq \bigcup_{t \in \mathcal{T}} [\mathbb{\Pi}_{\mathcal{I}}[\![proj_t(p)]\!]_t \langle \mathcal{E}_0, \emptyset, I \rangle]_\Omega$

Note: sound but not complete

Interference abstraction

Abstract interferences $\mathcal{I}^\sharp$

$\mathcal{P}(\mathcal{I}) \stackrel{\text{def}}{=} \mathcal{P}(\mathcal{T} \times \mathcal{V} \times \mathbb{R})$ is abstracted as $\mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathcal{V}) \to \mathcal{N}^\sharp$
where $\mathcal{N}^\sharp$ abstracts $\mathcal{P}(\mathbb{R})$ (e.g. intervals)

Abstract semantics with interferences $\mathbb{S}^\sharp_{\mathcal{I}}[\![s]\!]$

derived from $\mathbb{S}^\sharp$ in a generic way:

Example: $\mathbb{S}^\sharp_{\mathcal{I}}[\![X \leftarrow e]\!]_t \langle R^\sharp, \Omega, I^\sharp \rangle$

- for each Y in e, get its interference $Y^\sharp_{\mathcal{N}} = \bigsqcup^\sharp_{\mathcal{N}} \{ I^\sharp \langle t', Y \rangle \mid t' \neq t \}$
- if $Y^\sharp_{\mathcal{N}} \neq \bot^\sharp_{\mathcal{N}}$, replace Y in e with $get \langle Y, R^\sharp \rangle \sqcup^\sharp_{\mathcal{N}} Y^\sharp_{\mathcal{N}}$
- compute $\langle R^{\sharp\prime}, \Omega' \rangle = \mathbb{S}^\sharp[\![e]\!] \langle R^\sharp, \Omega \rangle$
- enriches $I^\sharp \langle t, X \rangle$ with $get(X, R^{\sharp\prime})$

Static analysis with interferences

Abstract analysis

$$\mathbb{P}^\sharp_{\mathcal{I}} \stackrel{\text{def}}{=} \left[\operatorname{lim} \lambda \langle\, \Omega,\ I^\sharp \,\rangle .\ \langle\, \Omega,\ I^\sharp \,\rangle \triangledown \bigsqcup^\sharp_{t \in \mathcal{T}} \left[\mathbb{S}^\sharp_{\mathcal{I}} [\![\, prog_t \,]\!]_t \langle\, \mathcal{E}^\sharp_0,\ \emptyset,\ I^\sharp \,\rangle \right]_{\Omega, I^\sharp} \right]_\Omega$$

- effective analysis by structural induction
- termination ensured by a widening
- parametrized by a choice of abstract domains $\mathcal{N}^\sharp$, $\mathcal{E}^\sharp$

- interferences are flow-insensitive and non-relational in $\mathcal{N}^\sharp$
- thread analysis remains flow-sensitive and relational in $\mathcal{E}^\sharp$

Weak memory consistency

Issues with weak consistency

program written

$\mathcal{E}_0 : F_1 = F_2 = 0$

$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
S_1	S_2

(simplified Dekker mutual exclusion algorithm)

S_1 and S_2 cannot execute simultaneously.

Issues with weak consistency

program written

$\mathcal{E}_0 : F_1 = F_2 = 0$

$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
S_1	S_2

$\longrightarrow$

program executed

$\mathcal{E}_0 : F_1 = F_2 = 0$

if $F_2 = 0$ **then**	**if** $F_1 = 0$ **then**
$F_1 \leftarrow 1;$	$F_2 \leftarrow 1;$
S_1	S_2

(simplified Dekker mutual exclusion algorithm)

S_1 and S_2 can execute simultaneously.
Not a sequentially consistent behavior!

Caused by:

- write FIFOs, caches, distributed memory
- hardware or compiler optimizations, transformations
- ...

behavior accepted by Java [Manson al. 05]

Out of thin air principle

original program

$\mathcal{E}_0 : R_1 = R_2 = X = Y = 0$

$R_1 \leftarrow X;$	$R_2 \leftarrow Y;$
$Y \leftarrow R_1$	$X \leftarrow R_2$

(example from causality test case #4 for Java by Pugh et al.)

We should not have $R_1 = 42$.

Out of thin air principle

original program

$\mathcal{E}_0 : R_1 = R_2 = X = Y = 0$	
$R_1 \leftarrow X;$	$R_2 \leftarrow Y;$
$Y \leftarrow R_1$	$X \leftarrow R_2$

$\longrightarrow$

"optimized" program

$\mathcal{E}_0 : R_1 = R_2 = X = Y = 0$	
$Y \leftarrow 42;$	
$R_1 \leftarrow X;$	$R_2 \leftarrow Y;$
$Y \leftarrow R_1$	$X \leftarrow R_2$

(example from causality test case #4 for Java by Pugh et al.)

We should not have $R_1 = 42$.

Possible if we allow speculative writes!
$\Longrightarrow$ we disallow this kind of program transformations.

(also forbidden in Java)

Path-based definition of weak consistency

Acceptable control path transformations: $p \rightsquigarrow q$

only reduce interferences and errors

- Reordering: $X_1 \leftarrow e_1 \cdot X_2 \leftarrow e_2 \rightsquigarrow X_2 \leftarrow e_2 \cdot X_1 \leftarrow e_1$
  (if $X_1 \notin var(e_2)$, $X_2 \notin var(e_1)$, and e_1 does not stop the program)
- Propagation: $X \leftarrow e \cdot s \rightsquigarrow X \leftarrow e \cdot s[e/X]$
  (if $X \notin var(e)$, $var(e)$ are thread-local, and e is deterministic)
- Factorization: $s_1 \cdot \ldots \cdot s_n \rightsquigarrow X \leftarrow e \cdot s_1[X/e] \cdot \ldots \cdot s_n[X/e]$
  (if X is fresh, $\forall i$, $var(e) \cap lval(s_i) = \emptyset$, and e has no error)
- Decomposition: $X \leftarrow e_1 + e_2 \rightsquigarrow T \leftarrow e_1 \cdot X \leftarrow T + e_2$
  (change of granularity)
- ...

but **NOT:**

- "out-of-thin-air" writes: $X \leftarrow e \rightsquigarrow X \leftarrow 42 \cdot X \leftarrow e$

Soundness of the interference semantics

Interleaving semantics of transformed programs $\mathbb{P}'_*$

- $\pi'(s) \stackrel{\text{def}}{=} \{ p \mid \exists p' \in \pi(s),\ p' \rightsquigarrow^* p \}$
- $\pi'_* \stackrel{\text{def}}{=} \{ \text{interleavings of } \pi'(prog_t),\ t \in \mathcal{T} \}$
- $\mathbb{P}'_* \stackrel{\text{def}}{=} [\, \sqcap [\![\pi'_*]\!] \langle \mathcal{E}_0, \emptyset \rangle \,]_\Omega$

Soundness theorem

$$\mathbb{P}'_* \subseteq \mathbb{P}_{\mathcal{I}}$$

$\Longrightarrow$ the interference semantics is sound wrt. weakly consistent memories

Synchronization

Scheduling

Synchronization primitives

$$
\begin{array}{rcl}
\mathit{stat} & ::= & \textbf{lock}(m) \\
& | & \textbf{unlock}(m) \\
& | & X \leftarrow \textbf{islocked}(m) \\
& | & \textbf{yield}
\end{array}
$$

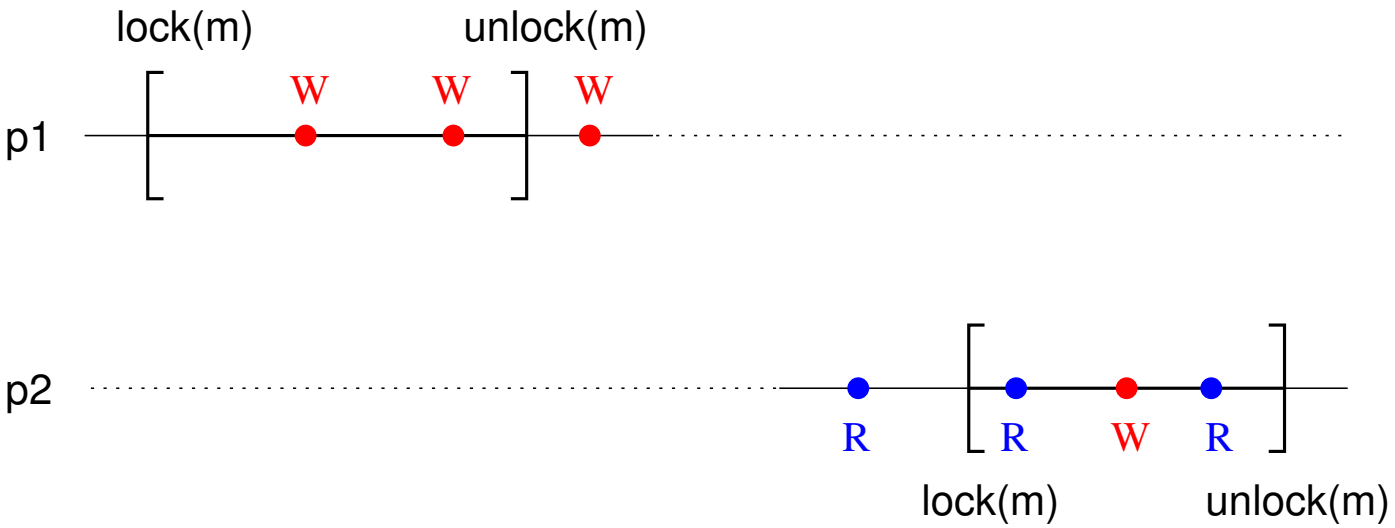
$m \in \mathcal{M}$: finite set of non-recursive mutexes

threads $t_1, \ldots, t_n$ have fixed, distinct priorities

Real-time scheduling

- only the highest priority unblocked thread can run
- **lock** and **yield** may block
- **yield**ing threads wake up non-deterministically (preempting lower-priority threads)
- explicit synchronisation enforces memory consistency

Mutual exclusion



Interleaving semantics $\mathbb{P}_*$:

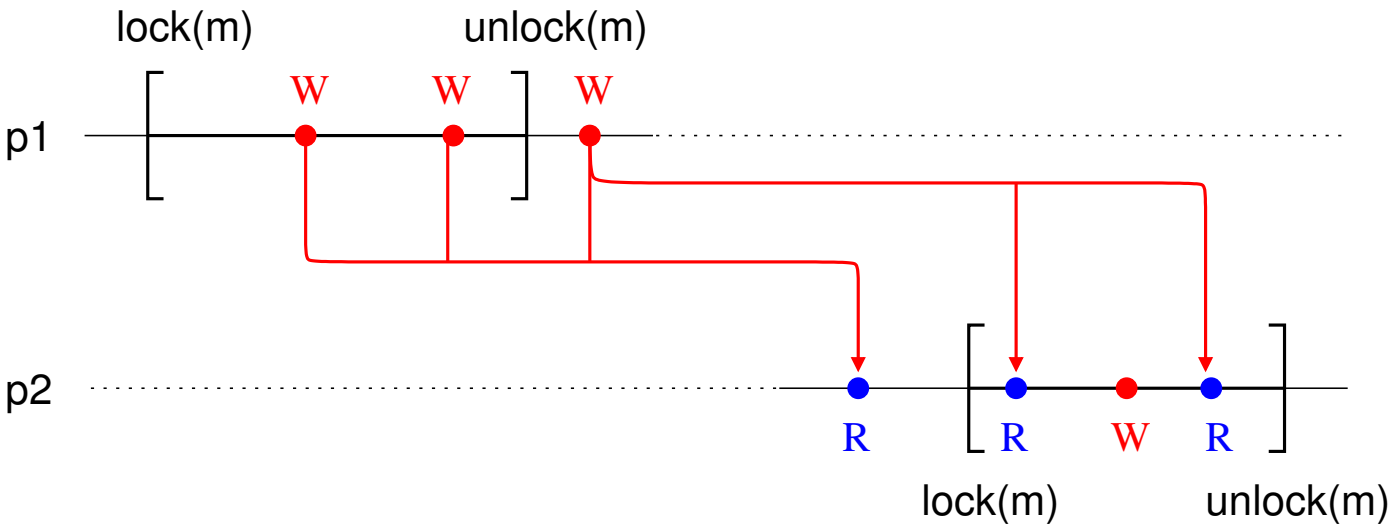
restrict interleavings of control paths

Interference semantics $\mathbb{P}_{\mathcal{I}}$, $\mathbb{P}^\sharp_{\mathcal{I}}$:

partition wrt. an abstract local view of the scheduler $\mathcal{C}$

- $\mathcal{E} \rightsquigarrow \mathcal{E} \times \mathcal{C}$, $\quad \mathcal{E}^\sharp \rightsquigarrow \mathcal{C} \to \mathcal{E}^\sharp$
- $\mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathcal{V} \times \mathbb{R} \rightsquigarrow \mathcal{I} \stackrel{\text{def}}{=} \mathcal{T} \times \mathcal{C} \times \mathcal{V} \times \mathbb{R}$,
  $\mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathcal{V}) \to \mathcal{N}^\sharp \rightsquigarrow \mathcal{I}^\sharp \stackrel{\text{def}}{=} (\mathcal{T} \times \mathcal{C} \times \mathcal{V}) \to \mathcal{N}^\sharp$

Mutual exclusion

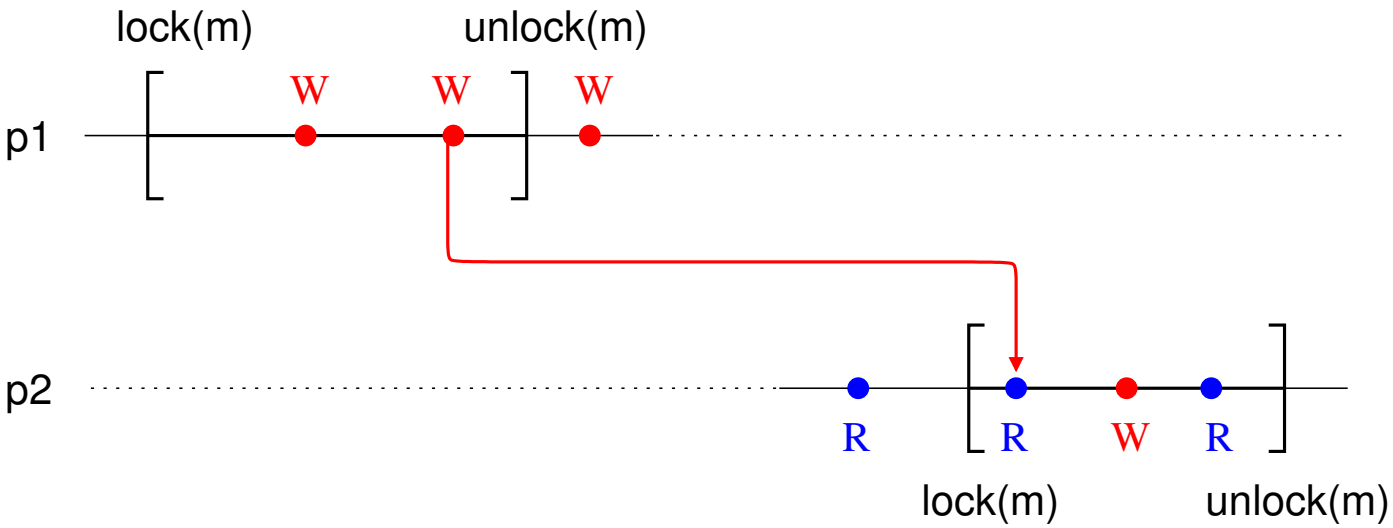


Data-race effects

Partition wrt. mutexes $M \subseteq \mathcal{M}$ held by the current thread t

- $\mathbb{S}_{\mathcal{I}}[\![X \leftarrow e]\!]_t \langle \rho, M, I \rangle$ adds
  $\{ \langle t, M, X, v \rangle \mid v \in \mathbb{E}_{\mathcal{I}}[\![X]\!]_t \langle \rho, M, I \rangle \}$ to I
- $\mathbb{E}_{\mathcal{I}}[\![X]\!]_t \langle \rho, M, I \rangle =$
  $\{ \rho(X) \} \cup \{ v \mid \langle t', M', X, v \rangle \in I,\ t \neq t',\ M \cap M' = \emptyset \}$
- flow-insensitive, subject to weak memory consistency

Mutual exclusion



Well-synchronized effects

- last write before unlock affects first read after lock
- partition interferences wrt. a protecting mutex m (and M)
- $\mathbb{S}_{\mathcal{I}}[\![\, \textbf{unlock}(m) \,]\!]_t \langle\, \rho,\, M,\, I \,\rangle$ stores all $\rho(X)$ into I
- $\mathbb{S}_{\mathcal{I}}[\![\, \textbf{lock}(m) \,]\!]_t \langle\, \rho,\, M,\, I \,\rangle$ imports values form I into ρ
- imprecision: non-relational, largely flow-insensitive

Example analysis

abstract consumer/producer

$$\mathcal{E}_0 : X = 1$$

p1:	p2:
while 1 **do**	**while** 1 **do**
lock(m);[1]	**lock**(m);
if $X > 0$ **then** [2] $X \leftarrow X - 1$;	$X \leftarrow X + 1$;
unlock(m);	**if** $X > 10$ **then** $X \leftarrow 10$;
[3] $Y \leftarrow X$	**unlock**(m)

- at [1] the **unlock** − **lock** effect from $p2$ imports $\{X\} \times [1, 10]$
- at [2] $X \in [1, 10]$, no effect from $p2$: $X \leftarrow X - 1$ is safe
- at [3] $X \in [0, 9]$, and $p2$ has the effects $\{X\} \times [1, 10]$
  so, $Y \in [0, 10]$

Real-time scheduling

priority-based critical sections

high thread	low thread
$L \leftarrow$ **islocked**(m);	**lock**(m);
if $L = 0$ **then**	$Z \leftarrow Y$;
$Y \leftarrow Y + 1$;	$Y \leftarrow 0$;
yield	**unlock**(m)

Partition interferences and memory states wrt. scheduling state

- partition wrt. mutexes tested with **islocked**
- $L \leftarrow$ **islocked**(m) creates two partitions
  - P_0 where $L = 0$ and m is free
  - P_1 where $L = 1$ and m is locked
- P_0 handled as if m where locked
- blocking primitives merge P_0 and P_1 (**lock**, **yield**)

AstréeA

AstréeA project

Goal: Astrée for asynchronous programs

Target programs: large embedded avionic C software

Model: ARINC 653 real-time operating system

- several concurrent threads, one a single processor
- shared memory (implicit communications)
- synchronisation primitives (mutexes)
- real-time scheduling (priority-based)
- fixed set of threads and mutexes, fixed priorities
- no dynamic memory allocation, no recursivity

We compute all run-time errors in a sound way:

- classic C run-time errors (overflows, invalid pointers, etc.)
- data-races (report & factor in the analysis)

but NOT deadlocks, livelocks, priority inversions (yet!)

The AstréeA prototype

Prototype:

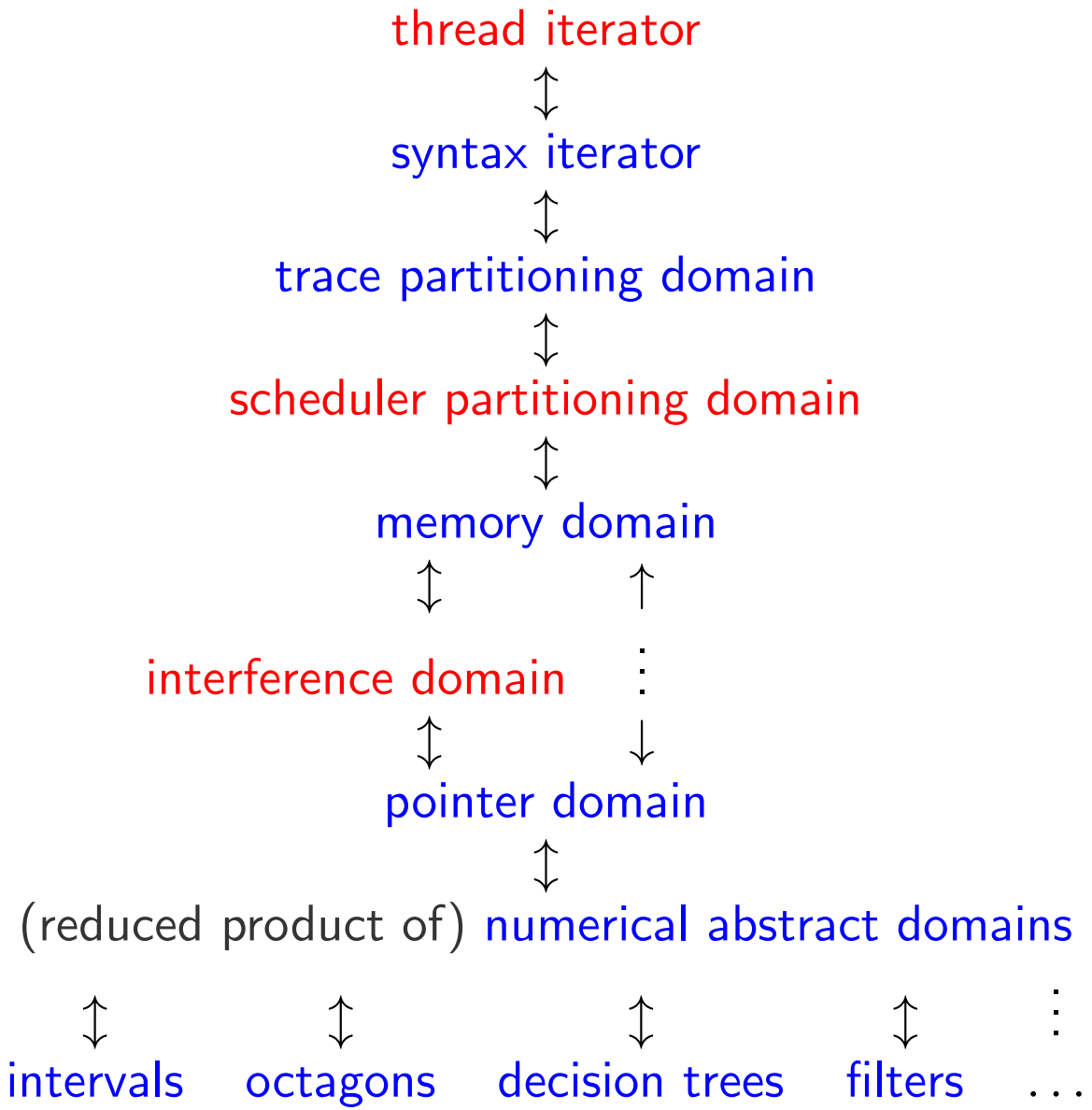
- started in 2009
- based on the Astrée code-base
- additions:
  - support for multithreaded programs
  - checks for data-races
  - new or improved abstract domains
  - added flexibility and scalability
  - improved iteration strategies

Website

http://www.astreea.ens.fr

work in progress

Abstract interpreter



Target system

- embedded avionic code
- 1.6 Mlocs of C, 15 threads
- many variables, large arrays, many loops
- reactive code + network code + lists, strings. pointers
- initialization phase, followed by a multithreaded phase

Target system

Operating system:

- ARINC 653 OS specification
- model hand-written based on documentation in C + low-level analyzer primitives (2.6 Klines)

Memory consistency: conservative model

Analysis results

Analysis on our intel 64-bit 2.66 GHz server, 64 GB RAM.

Analysis results

lines	# threads	# iters.	time	# alarms
100 K	5	4	46 mn	64
1.6 M	15	6	43 h	1 208

efficiency on par with analyses of synchronous code

- few thread reanalyses
- few partitions (up to 4 for memory states, 52 for interferences)

but still too many alarms

Limitations of the interference abstraction

Lack of relational lock invariants

a difficult example

$$\mathcal{E}_0 : X = Y = 5$$

while 1 **do**	**while** 1 **do**
lock(m);	**lock**(m);
if $X > 0$ **then**	**if** $X < 10$ **then**
$X \leftarrow X - 1;$	$X \leftarrow X + 1;$
$Y \leftarrow Y - 1;$	$Y \leftarrow Y + 1;$
unlock(m)	**unlock**(m)

Our analysis finds $X \in [0, 10]$, but no bound on Y.

Actually $Y \in [0, 10]$.

To prove this, we would need to infer the relational invariant $X = Y$ at lock boundaries.

Lack of inter-process flow-sensitivity

a more difficult example

$\mathcal{E}_0 : X = 0$

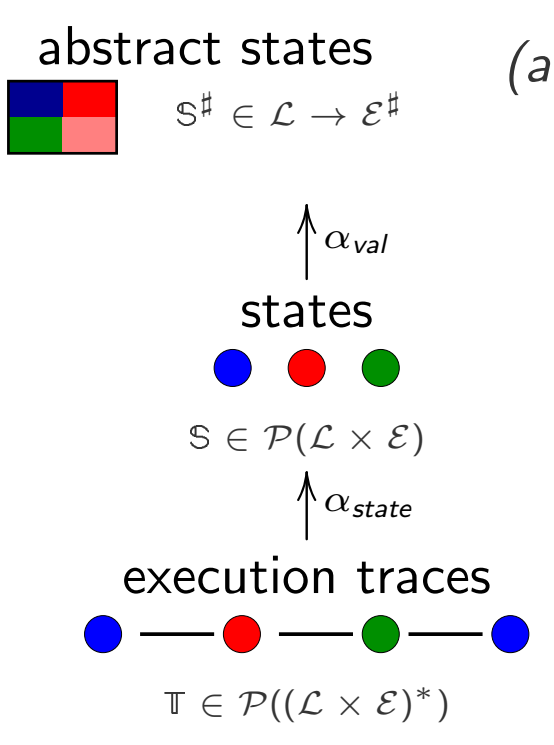
while 1 **do**	**while** 1 **do**
lock(m);	**lock**(m);
$X \leftarrow X + 1$;	$X \leftarrow X + 1$;
unlock(m);	**unlock**(m);
lock(m);	**lock**(m);
$X \leftarrow X - 1$;	$X \leftarrow X - 1$;
unlock(m)	**unlock**(m)

Our analysis finds no bound on X.

Actually $X \in [-2, 2]$ at all program points.
To prove this we need to infer an invariant on the history of interleaved executions:
no more than two incrementation (resp. decrementation) can occur without a decrementation (resp. incrementation).

Towards more general interferences

Reminder: from traces to sequential analyses



(abstract invariants)

(invariants)

implementable
data-structures + algorithms

mathematical
non-computable

- $\mathbb{T} \stackrel{\text{def}}{=} \text{lfp}\, \lambda T.\, I \cup \{ \langle \sigma_0, \ldots, \sigma_{n+1} \rangle \mid \langle \sigma_0, \ldots, \sigma_n \rangle \in T \wedge \sigma_n \to \sigma_{n+1} \}$
- $\mathbb{S} \stackrel{\text{def}}{=} \text{lfp}\, \lambda S.\, I \cup \{ \sigma \mid \exists \sigma' \in S \wedge \sigma' \to \sigma \} = \alpha_{state}(\mathbb{T})$
  where $\alpha_{state}(T) \stackrel{\text{def}}{=} \{ \sigma_i \mid \exists \langle \sigma_0, \ldots, \sigma_n \rangle \in T \wedge i \in [0, n] \}$

Interleaved traces

States:

- thread state: for each $t \in \mathcal{T}$, $\Sigma_t \stackrel{\text{def}}{=} \mathcal{L}_t \times \mathcal{E}$
- program state: $\Sigma \stackrel{\text{def}}{=} (\prod_{t \in \mathcal{T}} \mathcal{L}_t) \times \mathcal{E}$

Labelled transition relation: $\dot{\rightarrow} \in \Sigma \times \mathcal{T} \times \Sigma$

Labelled trace semantics:

$\mathbb{T} \stackrel{\text{def}}{=}$ lfp F where

$$F(T) \stackrel{\text{def}}{=} I \cup \{ \sigma_0 \stackrel{t_0}{\rightarrow} \cdots \stackrel{t_i}{\rightarrow} \sigma_{i+1} \mid \sigma_0 \stackrel{t_0}{\rightarrow} \cdots \stackrel{t_{i-1}}{\rightarrow} \sigma_i \in T \wedge \sigma_i \stackrel{t_i}{\rightarrow} \sigma_{i+1} \}$$

Interleaved trace abstraction

Complementary abstractions:

- thread-local states: $\mathbb{S}_t \stackrel{\text{def}}{=} \pi_t(\alpha_{state}(\mathbb{T})) \in \mathcal{P}(\mathcal{L}_t \times \Sigma')$

  $\pi_t(\ell, \rho) \stackrel{\text{def}}{=} (\ell_t,\ \rho[\forall t' \neq t,\ pc_{t'} \mapsto \ell_{t'}])$

  (inter-thread flow-sensitive state, with auxiliary variables)

- interferences: $\mathbb{A}_t \stackrel{\text{def}}{=} \{\, (\sigma_i, \sigma_{i+1}) \,|\, \exists \cdots \sigma_i \xrightarrow{t} \sigma_{i+1} \cdots \in \mathbb{T} \,\}$

  (relational flow-sensitive interferences)

Nested fixpoint form:

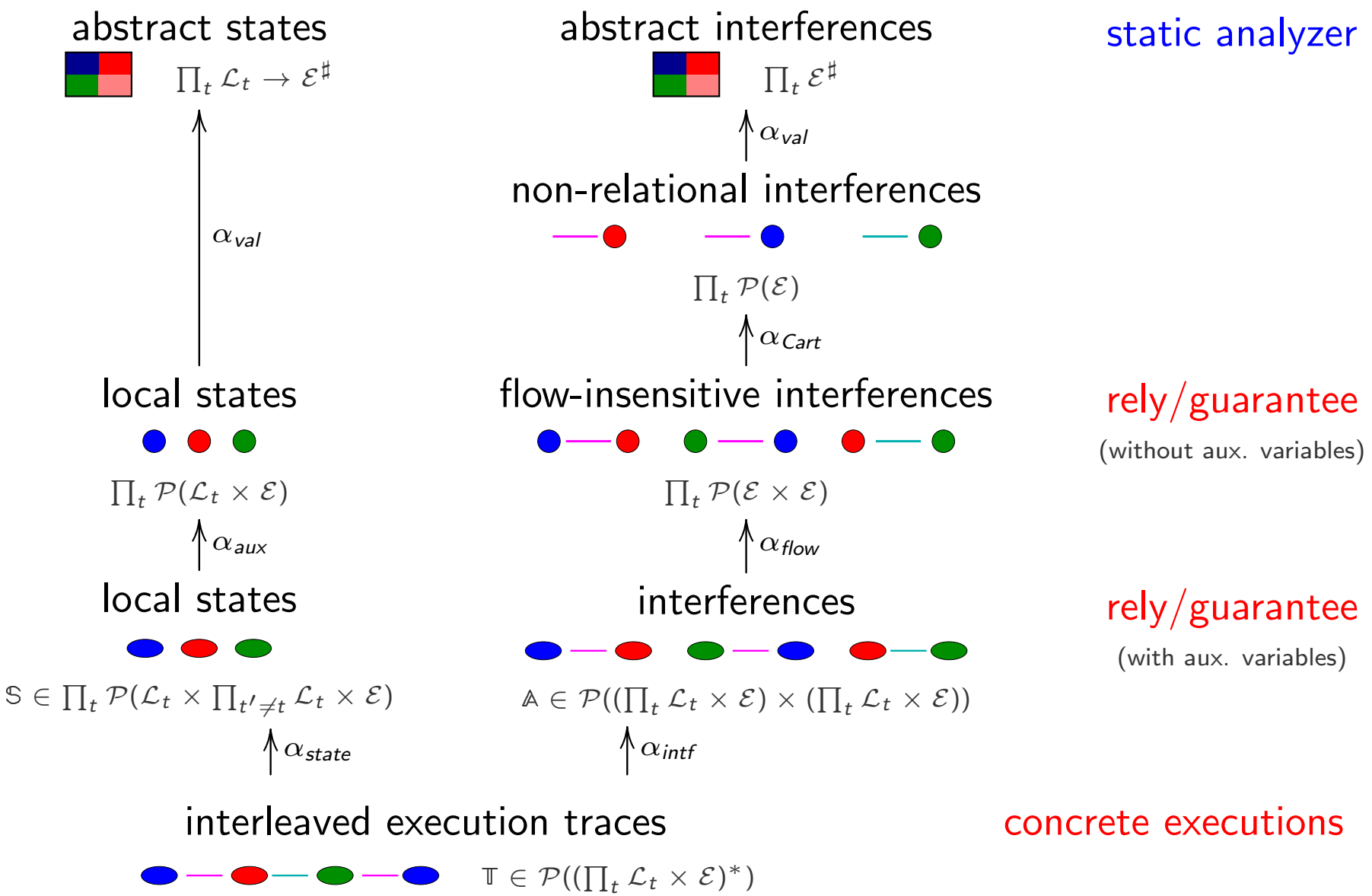
$\mathbb{S}_t = \text{lfp}\ G_t$ where

$G_t(S) \stackrel{\text{def}}{=} \text{lfp}\ H_t(\lambda t'.\, \{\, (\sigma, \sigma') \,|\, \sigma \in S_{t'},\ \sigma \xrightarrow{t'} \sigma' \,\})$

$H_t(A)(S) \stackrel{\text{def}}{=} \pi_t(I \cup \{\, \sigma' \,|\, \exists \pi_t(\sigma) \in S,\ \sigma \xrightarrow{t} \sigma' \vee \exists t' \neq t, (\sigma, \sigma') \in A_{t'} \,\})$

The abstraction $\mathbb{S}$ is complete for safety properties.

From traces to thread-modular analyses



Conclusion

Conclusion

A method to analyze embedded real-time multithreaded programs that:

- is sound for all interleavings
- is sound for weakly consistent memory semantics
- takes scheduling and synchronization into account
- is parametrized by abstract domains
- can benefit directly from existing non-parallel analyzers
- is efficient (on par with non-parallel analysis)

Encouraging experimental results.

Related work

Huge body of work on parallel programs, models & analysis!

Main inspirations:

- **proofs** of parallel programs [OwickiGries76, Lamport77]
  decomposition: thread-local invariant + global interference
- **abstract interpretation**
  formalizing the proof methods [CousotCousot84]
  recent static analysis applications [Ferrara08, CarréHymans09]

Other works:

- **flow-insensitive analyses** [Steensgaard96]
  naturally handle parallelism, but low precision
- **model-checking**: considers interleavings explicitly
  partial-order reduction (complete) [Godefroid94]
  bounded context switches (unsound) [QadeerRehof05]
- **weak memory consistency**
  hardware [Lamport79, AdveGharachorloo96]
  language [MansonPughAdve05] (Java), [SaraswatAI07] (generative)

Future work

Goal: zero alarm

Planned improvements:

- relational analysis of interferences
  (well-synchronized effects protected by mutexes)
- history-sensitive abstractions (ordering of critical sections)
- additional synchronisation primitives (events)
- additional priority policies (priority ceiling)
- other operating systems (OSEK/VDX, AUTOSAR, POSIX)
- refined weakly consistent memory models (atomic volatile)
- application-specific abstract domains (lists, strings)

Questions?

Want to work on AstréeA?

Post-doc positions available!