Static analysis of avionics software for security

TLS-SEC

David Delmas

Airbus Avionics Software

11 December 2018

based on slides by © P. Cousot, A. Miné, and X. Rival
École normale supérieure, New York University, Université P-M Curie

AIRBUS

Agenda

Airbus Cockpit Avionics Software

Software functions

Aircraft Control Domain (**C**, asm)

- flight control systems — DAL A
- flight warning systems — DAL B or DAL C
- communication systems — DAL C

Airline Information Services Domain (Java)

- administrative functions — DAL E
- maintenance support — DAL E

Software platforms

ATSU	LynxOS®-based *POSIX* Host Platform	x86, DAL C
IMA	*ARINC 653* Integrated Modular Avionics	PowerPC, DAL A
ASF	PikeOS®-based Avionics Server Function	PowerPC, DAL C

Safety standards

Critical avionics software is subject to certification:

- regulated by international standards (DO-178 rev. B/C)
- verification = more than half of the development cost
- mostly based on massive test campaigns & intellectual reviews

Current trend:

use of **formal methods** now acknowledged (DO-178C, DO-333)

- at the binary level, to replace testing
- at the source level, to replace intellectual reviews
- at the source level, to replace testing
  provided the correspondence with the binary is also certified

Agenda

Security concerns for aircraft embedded systems

Aircraft functions increasingly performed by software-intensive systems

Airline flight ops & cabin facilities → connectivity

1. to external untrusted networks
2. to avionics maintenance & ground-board communication

Security concerns for aircraft embedded systems

Aircraft functions i

Airline flight ops & cabin facilities → connectivity

1. to external untrusted networks
2. to avionics maintenance & ground-board communication

Navigation Surveillance & Air Traffic functions

(NOT safety-critical)

Controller-Pilot communication (CPDLC)

Surveillance out of radar coverage (ADS)

ACARS/ATN data-link protocols over radio or satellite

(no cryptographic authentication or secrecy)

expected traffic growth

→ automated trajectory modification and negotiation needed in the near future

→ increased focus on datalink security

Security concerns for aircraft embedded systems

Aircraft functions increasingly performed by software-intensive systems

Airline flight ops & cabin facilities → connectivity

1. to external untrusted networks
2. to avionics maintenance & ground-board communication

Enhanced control automation, HMI comfort, systems interoperability, configurability

→ exponential complexity increase (SW, HW, networks)

→ harder to ensure freedom from security vulnerabilities

Security concerns for aircraft embedded systems

Aircraft functions increasingly performed by software-intensive systems

Airline flight ops & cabin facilities → connectivity

1. to external untrusted networks
2. to avionics maintenance & ground-board communication

Enhanced control automation, HMI comfort, systems interoperability, configurability

→ exponential complexity increase (SW, HW, networks)

→ harder to ensure freedom from security vulnerabilities

Major trends in avionics

1. more and more generic avionics platforms
2. growing parts developed by third parties

Security risks for aircraft embedded systems

Major trends in avionics

More and more generic avionics platforms

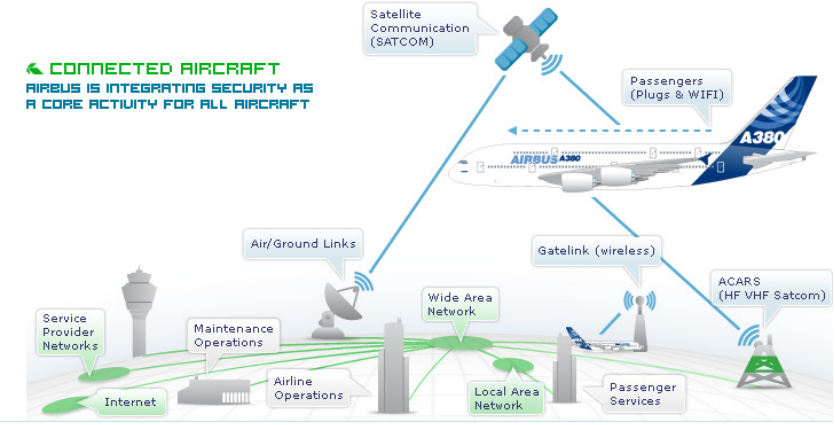
- → standard HW & communication protocols
- → *Commercial-Off-The-Shelf* components
- → likely to be known to unspecialised attackers

Growing parts developed by third parties

- → less grip on processes
- → need for automated security assurance

Airbus assets to be protected against threats

1. aircraft operational safety
2. aircraft operational & dispatch reliability
3. commercial branding & image (Airbus & Airline)



Increasing security-related regulatory constraints

Federal Register: April 13, 2007

`http://www.gpo.gov/fdsys/pkg/FR-2007-04-13/html/E7-7065.htm`

FAA emitted special conditions for **B787** certification bc

"The architecture of the Boeing Model 787-8 computer systems and networks may allow access to external systems and networks, such as wireless airline operations and maintenance systems, satellite communications, electronic mail, the internet, etc. Onboard wired and wireless devices may also have access to parts of the airplane's digital systems that provide flight critical functions." "These new connectivity capabilities may result in security vulnerabilities to the airplane's critical systems."

January 2010: similar special conditions for **B747-8**

`http://www.gpo.gov/fdsys/pkg/FR-2010-01-15/html/2010-661.htm`

December 2013: similar special conditions for **Airbus A350**

`https://federalregister.gov/a/2013-29985`

December 2013: EASA requested all airborne systems with datalink capabilities to be re-assessed for potential security vulnerabilities related to the processing of misformatted messages.

Pending standards for avionics

- RTCA Special Committee 216 (SC-216) to

  "help ensure safe, secure and efficient operations amid the growing use of highly integrated electronic systems and network technologies used on-board aircraft, for CNS/ATM systems and air carrier operations and maintenance."

- RTCA SC-216 / EUROCAE WG-72 collaboration
  1. *Minimum Aviation System Performance Standards (MASPS) for Aeronautical Electronic and Networked Systems Security*
  2. *Security Assurance and Assessment Processes and Methods for Safety-related Aircraft Systems*
  3. ED-202 *Airworthiness security process specification*
  4. ED-203 *Airworthiness Security Methodology and Instructions*

- State-of-the-art techniques should be proposed

Categories of security properties

Integrity

assets modified only by authorised parties in authorised ways

threats: buffer overflow attacks, spoofing

protections: access control, digital signatures, replication

Categories of security properties

Integrity

assets modified only by authorised parties in authorised ways

threats: buffer overflow attacks, spoofing

protections: access control, digital signatures, replication

Confidentiality

assets read only by authorised parties

threats: leakage of sensitive secret information

protections: ciphering, partitioning, information flow control

Categories of security properties

Integrity

assets modified only by authorised parties in authorised ways

threats: buffer overflow attacks, spoofing

protections: access control, digital signatures, replication

Confidentiality

assets read only by authorised parties

threats: leakage of sensitive secret information

protections: ciphering, partitioning, information flow control

Availability

assets are accessible to authorised parties in a timely manner

threats: denial of service attacks

Research directions

Integrity

- buffer overflow analyses (synchronous and asynchronous)
- information flow analyses (trusted vs tainted)
- formal proof of security functionalities

Confidentiality

- information flow analyses (public vs secret) (also integrity)
- formal verification of cryptographic protocols (also integrity)
- quantitative assessment of crypto primitives (also integrity)

Availability

- WCET and termination analyses
- resource consumption analyses

Main short-term objective: memory safety

Minimal integrity property

Goal: no run-time error!

- no buffer overflow, invalid pointer arithmetic or dereference
- no overflows in float, integer, `enum` arithmetic and cast
- no division, modulo by 0 on integers and floats

Main short-term objective: memory safety

Minimal integrity property

Goal: no run-time error!

- no buffer overflow, invalid pointer arithmetic or dereference
- no overflows in float, integer, `enum` arithmetic and cast
- no division, modulo by 0 on integers and floats

Means of assurance

- certified avionics $\Rightarrow$ soundness required
- internal software $\Rightarrow$ **C** source code available
- intellectual reviews costly and error-prone $\Rightarrow$ automation necessary

$\Rightarrow$ Solution: sound static analysis

Example: Sendmail buffer overflow

()()()...()

email address parsing

Discovered 2003 by Mark Dowd

```c
#define BUFFERSIZE 200
#define TRUE 1
#define FALSE 0
int copy_it ( char * input , unsigned int length ) {
    char c , localbuf [ BUFFERSIZE ];
    unsigned int upperlimit, quotation, roundquote, inputIndex, outputIndex;
    upperlimit = BUFFERSIZE - 10;
    quotation = roundquote = FALSE ;
    inputIndex = outputIndex = 0;
    while ( inputIndex < length ) {
        c = input [ inputIndex ++];
        if (( c == '<') && (! quotation )) {
            quotation = TRUE ; upperlimit --;
        }
        if (( c == '>') && ( quotation )) {
            quotation = FALSE ; upperlimit ++;
        }
        if (( c == '(') && (! quotation ) && ! roundquote ) {
            roundquote = TRUE ; //upperlimit--; // decrementation was missing
in bug
        }
        if (( c == ')') && (! quotation ) && roundquote ) {
            roundquote = FALSE ; upperlimit ++;
        }
        // If there is sufficient space in the buffer , write the character .
        if ( outputIndex < upperlimit ) {
            localbuf [ outputIndex ] = c;
            outputIndex ++;
        }
    }
    if ( roundquote ) {
        localbuf [ outputIndex ] = ')'; outputIndex ++; }
    if ( quotation ) {
        localbuf [ outputIndex ] = '>'; outputIndex ++; }

    return 0;
```

Agenda

Sound static analysis

Characteristics:

- direct analysis of the source code (not a model)
- automatic (easy to set up, no interaction with the user)
- efficient
- approximate (to sidestep decidability and efficiency issues)

Soundness:

- semantic-based (C specification, machine integers, floats, pointers, ...)
- full coverage of all control and data
- any property found by analysis holds on every program execution
  $\Longrightarrow$ no error missed, no false negative
- soundness is required by DO

Abstract interpretation: theory of the approximation of semantics
derive sound analysis with controllable cost/precision trade-off

Agenda

AIRBUS

Principle of abstract interpretation

Define the concrete semantics of your system or program

concrete semantics $\equiv$ mathematical model of the set of all its possible behaviours in all possible environments

can be constructed from semantics of commands of system specification or programming language

Principle of abstract interpretation

Define the concrete semantics of your system or program

concrete semantics ≡ mathematical model of the set of all its possible behaviours in all possible environments

can be constructed from semantics of commands of system specification or programming language

Define a specification

specification ≡ subset of possible behaviours

Principle of abstract interpretation

Define the concrete semantics of your system or program

concrete semantics ≡ mathematical model of the set of all its possible behaviours in all possible environments

can be constructed from semantics of commands of system specification or programming language

Define a specification

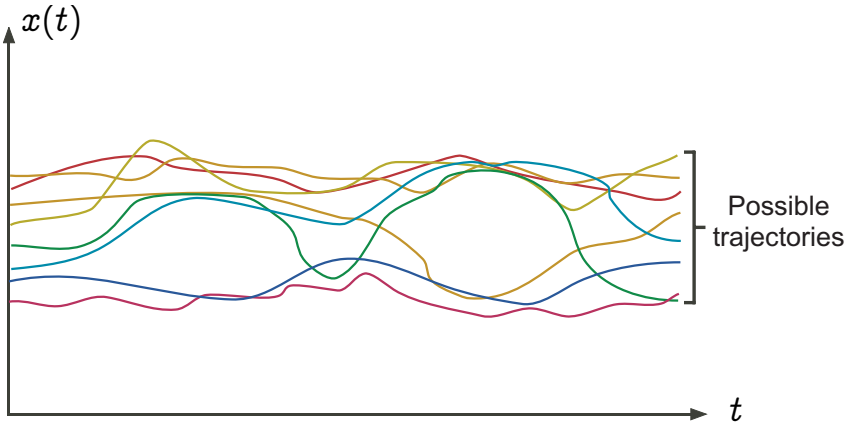
specification ≡ subset of possible behaviours

Conduct a formal proof

that the concrete semantics meets the specification

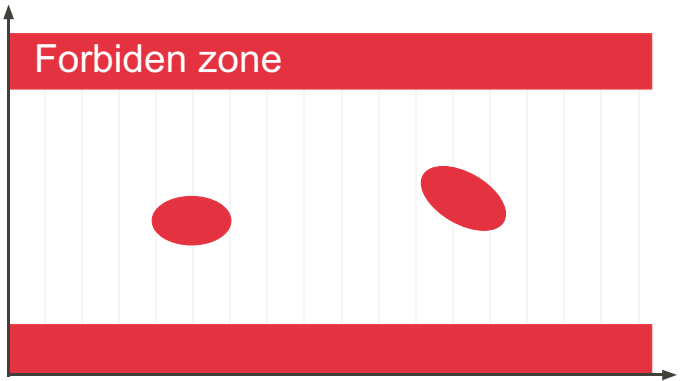
use computers to automate the proof

Concrete semantics of system/program P



$Semantics[|P|]$

Specification of P (e.g. safety property)



$Specification[|P|]$

Formal proof of P

$$Semantics[|P|] \subseteq Specification[|P|]$$

Excluded miracle

Undecidability

The concrete semantics of a system/program is not computable.

$\Rightarrow$ Most questions on system/program behaviour are undecidable.

Excluded miracle

Undecidability

The concrete semantics of a system/program is not computable.

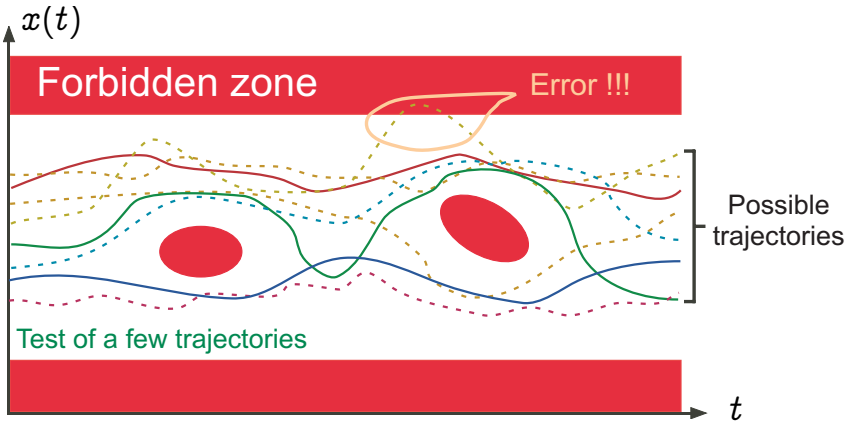
$\Rightarrow$ Most questions on system/program behaviour are undecidable.

Example: termination is undecidable

- assume `termination(P)` always terminates and returns `true` iff `P` always terminates on all input data
- the following program yields a contradiction

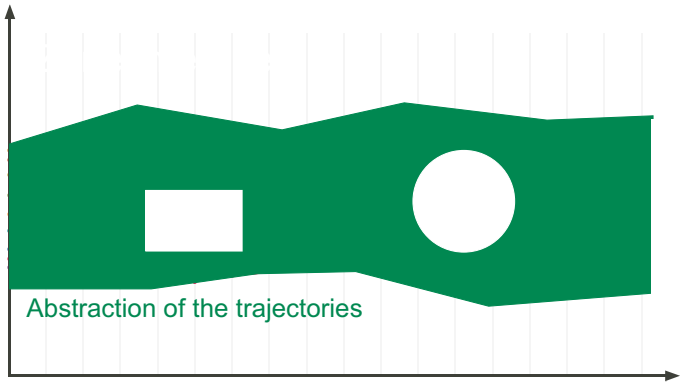
```
P := while termination(P)do ()done
```

Test/simulation



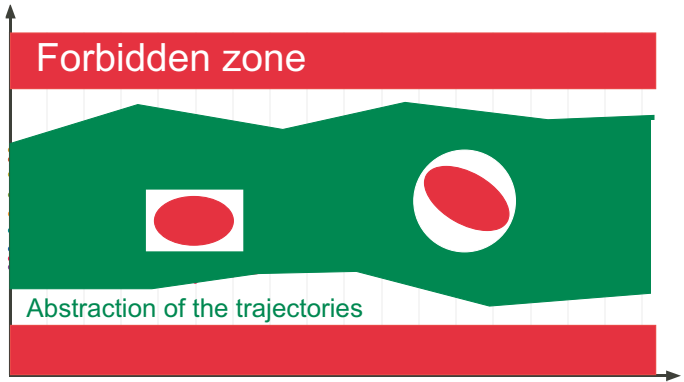
Subset of the possible behaviours $\Rightarrow$ incomplete

Abstract semantics for P



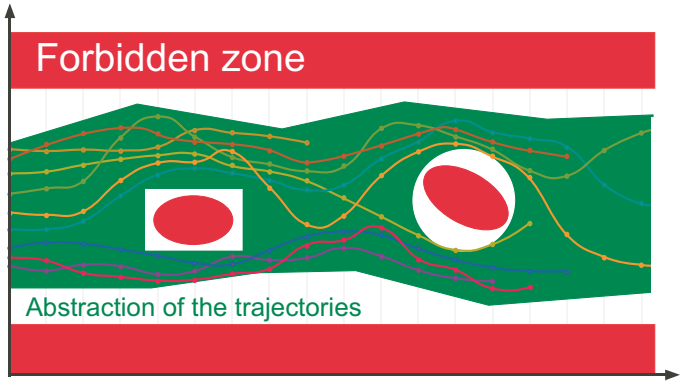
$$Abstraction(Semantics[\![P]\!])$$

Proof by abstract interpretation



$$Abstraction(Semantics[|P|]) \subseteq Specification[|P|]$$

Soundness of abstract interpretation



$$Semantics[|P|] \subseteq Abstraction(Semantics[|P|]) \subseteq Specification[|P|]$$

Formal methods are abstract interpretations

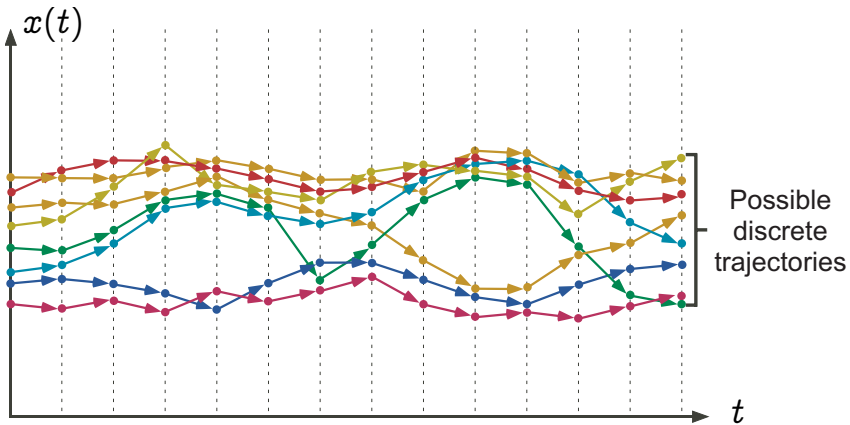
model-checking user-provided abstract semantics
$\triangleq$ finitary model
may be inferred by static analysis

deductive methods user-provided abstract semantics
$\triangleq$ inductive invariants
may be inferred by static analysis

static analysis abstract semantics computed automatically
$\triangleq$ predefined abstractions
may be tailored by the user

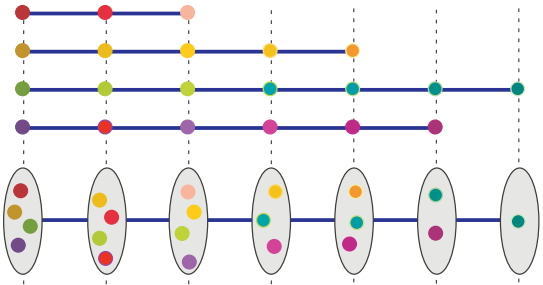
Agenda

Concrete semantics: set of (discrete) traces



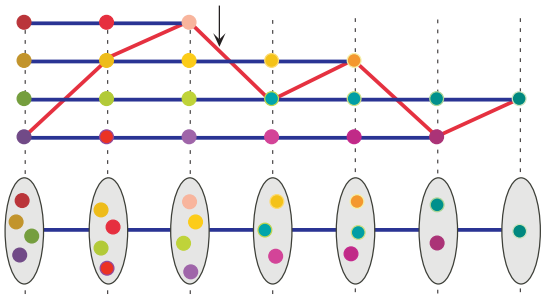
Collecting semantics

Collect the set of states that can appear on some trace at any given discrete time :

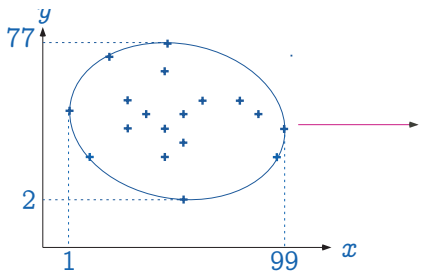


Trace abstraction : collecting abstraction

This an abstraction. Does the red trace exist?
Trace semantics: no $\neq$ collecting semantics: I don't know.

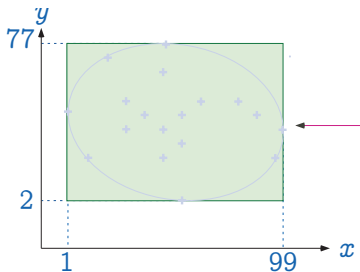


Set abstraction : intervals



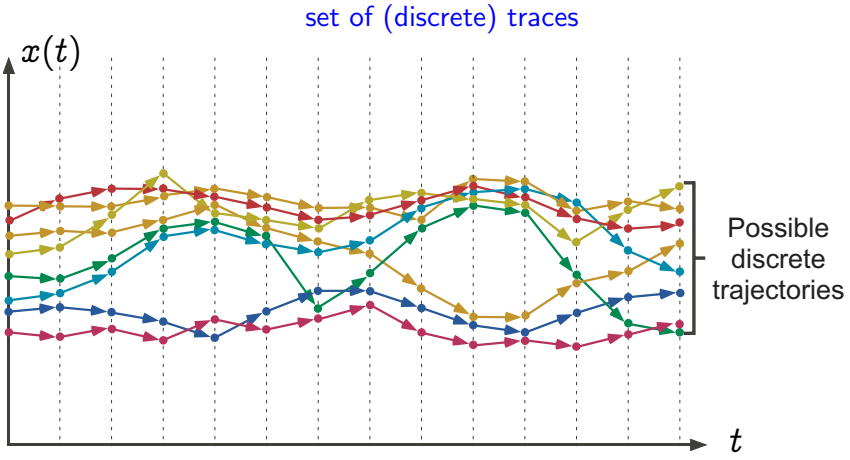
$\{x : [1, 99], y : [2, 77]\}$

Set abstraction : intervals

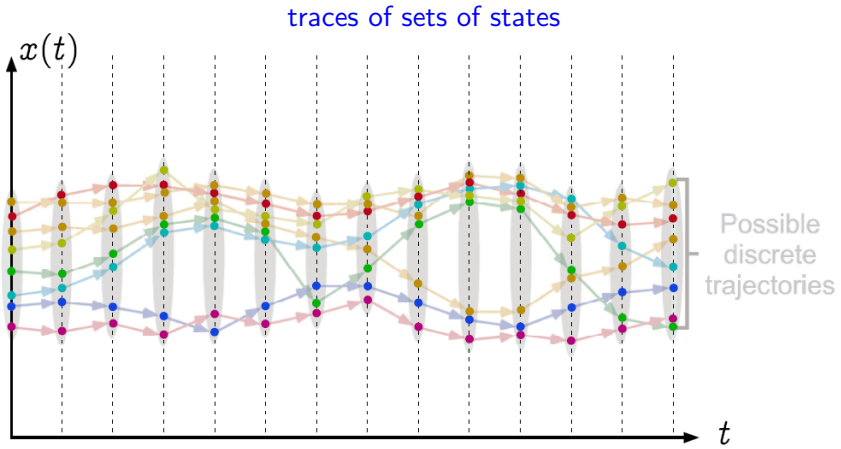


$\{x : [1, 99], y : [2, 77]\}$

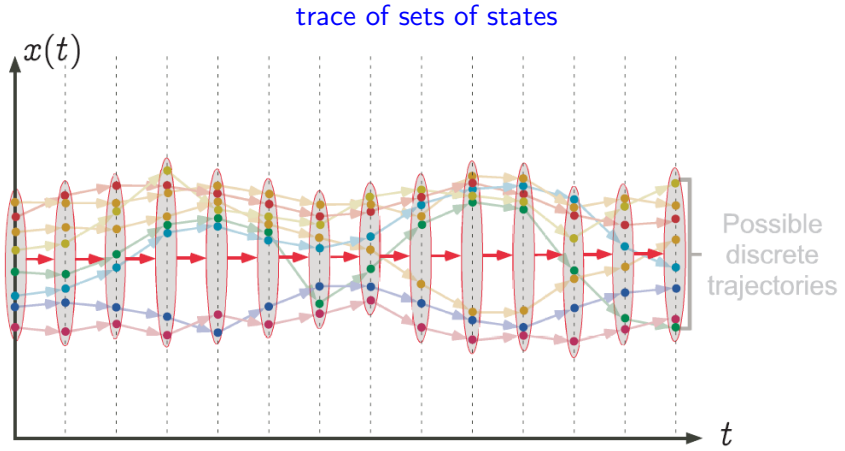
From set of traces to set of reachable states



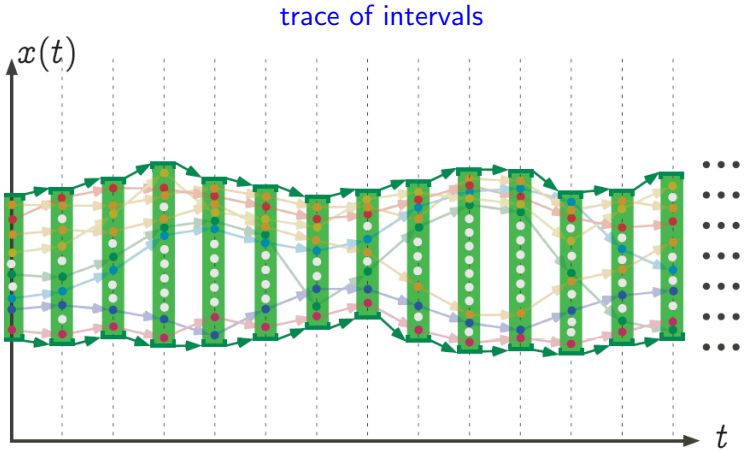
From set of traces to set of reachable states



From set of traces to set of reachable states

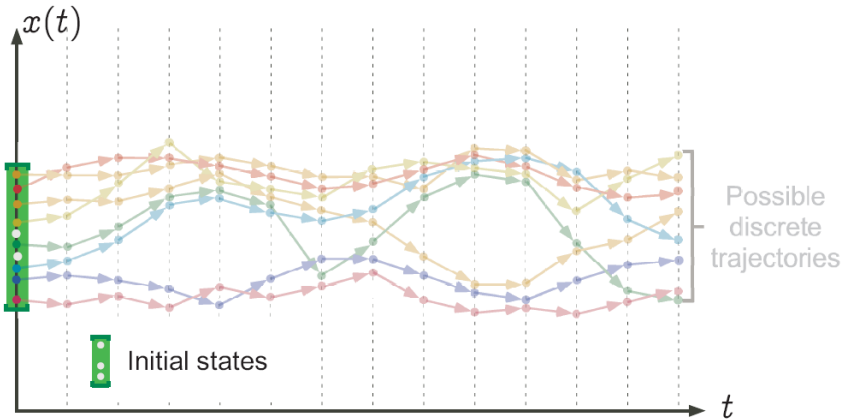


From set of traces to set of reachable states



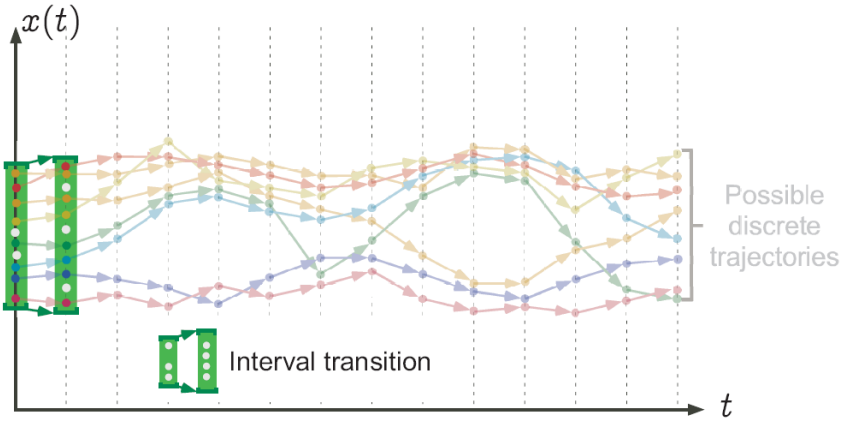
From set of traces to set of reachable states

Effective computation : intialisation



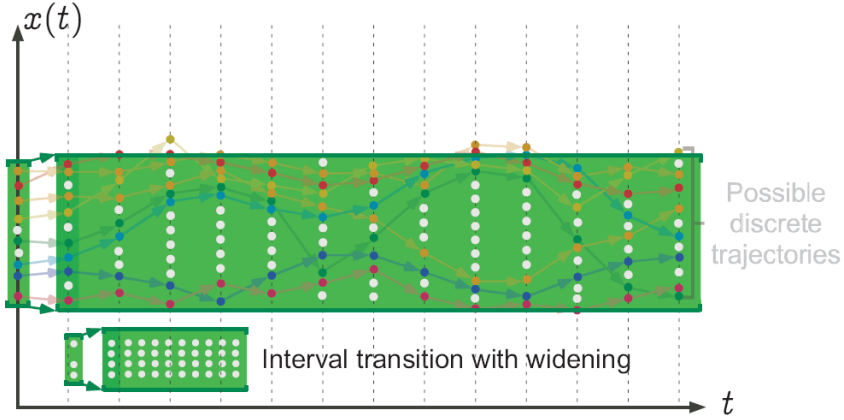
From set of traces to set of reachable states

Effective computation : propagation



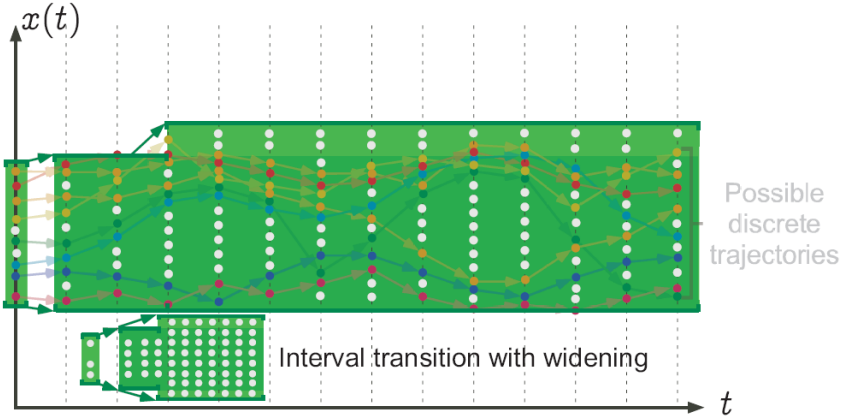
From set of traces to set of reachable states

Effective computation : widening unstable constraints



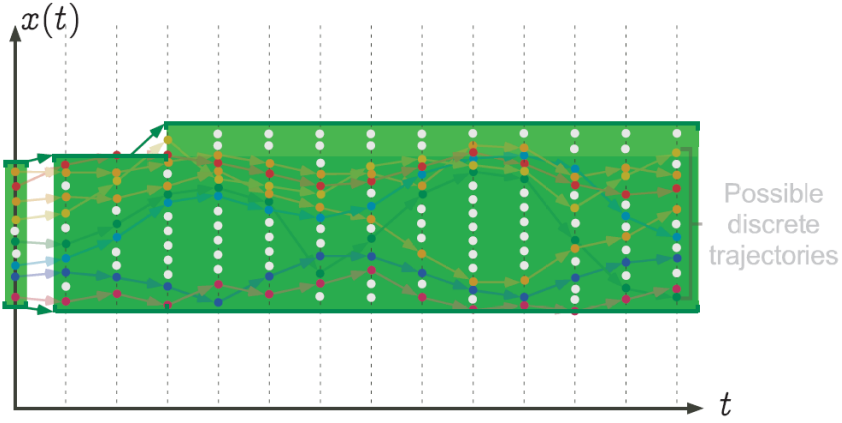
From set of traces to set of reachable states

Effective computation : widening unstable constraints



From set of traces to set of reachable states

Effective computation : stability of interval constraints



Interval analysis

Program to be analyzed

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

Interval analysis

Equations (abstract interpretation of the semantics)

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = \emptyset \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = \emptyset \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = \emptyset \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,1] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
        x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,2] \\ X_3 = [2,2] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,2] \\ X_3 = [2,3] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,3] \\ X_3 = [2,3] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,3] \\ X_3 = [2,4] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,4] \\ X_3 = [2,4] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,4] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,5] \\ X_3 = [2,5] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Resolution by increasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,5] \\ X_3 = [2,6] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Convergence speed-up by widening

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, +\infty] \quad \Leftarrow \text{widening} \\ X_3 = [2,6] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Decreasing iterations

```
    x := 1;
1:
    while x < 10000 do
2:
        x := x + 1
3:
    od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1, +\infty] \\ X_3 = [2, +\infty] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Decreasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
        x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,9999] \\ X_3 = [2,+\infty] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Decreasing iterations

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,9999] \\ X_3 = [2,+10000] \\ X_4 = \emptyset \end{cases}$$

Interval analysis

Final solution

```
   x := 1;
1:
   while x < 10000 do
2:
       x := x + 1
3:
   od;
4:
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,9999] \\ X_3 = [2,+10000] \\ X_4 = [+10000,+10000] \end{cases}$$

Interval analysis

Result of interval analysis

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}
       x := x + 1
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

$$\begin{cases} X_1 = [1,1] \\ X_2 = (X_1 \cup X_3) \cap [-\infty, 9999] \\ X_3 = X_2 \oplus [1,1] \\ X_4 = (X_1 \cup X_3) \cap [10000, +\infty] \end{cases}$$

$$\begin{cases} X_1 = [1,1] \\ X_2 = [1,9999] \\ X_3 = [2,+10000] \\ X_4 = [+10000,+10000] \end{cases}$$

Interval analysis

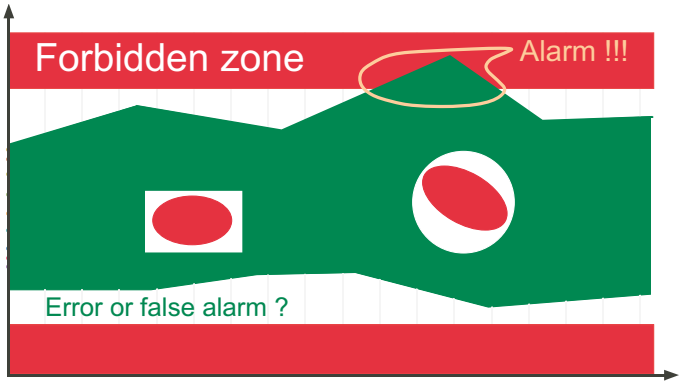
Formal proof of absence of overflow

```
    x := 1;
1: {x = 1}
    while x < 10000 do
2: {x ∈ [1, 9999]}
        x := x + 1
3: {x ∈ [2, +10000]}
    od;
4: {x = 10000}
```

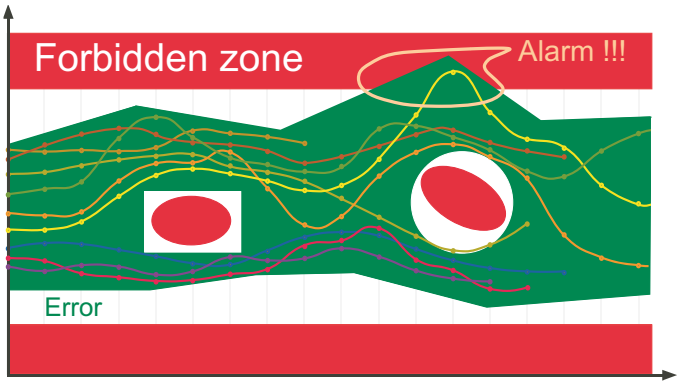
⟵ no overflow

Agenda

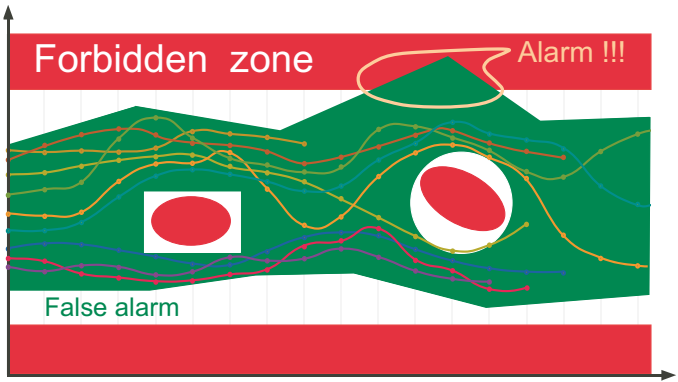
Alarms



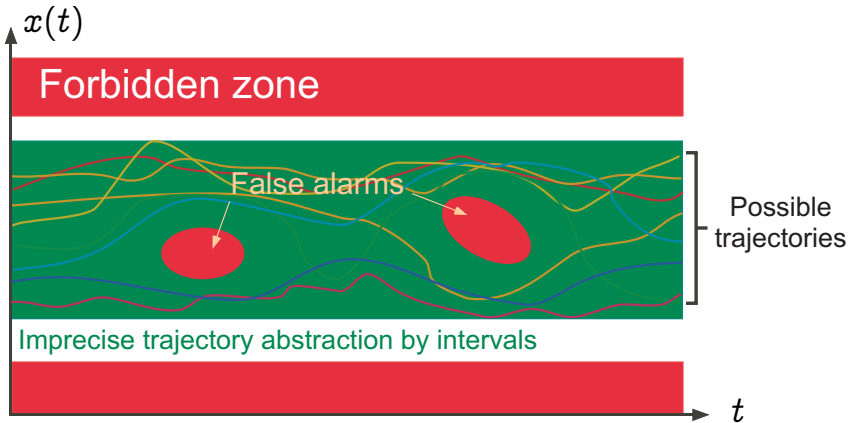
True error



Incompleteness ⇒ false alarms



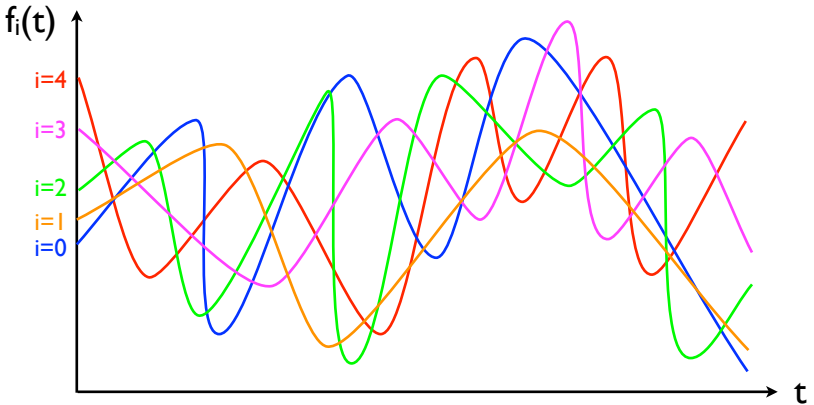
False alarms (e.g. interval analysis)



$\Rightarrow$ need for abstraction refinement

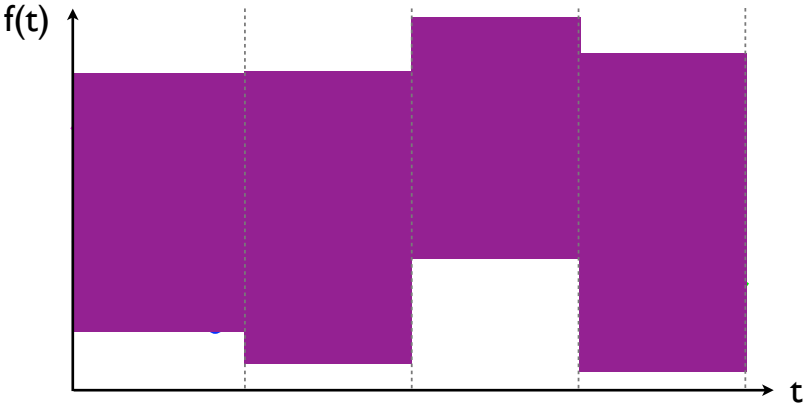
Agenda

Set of functions

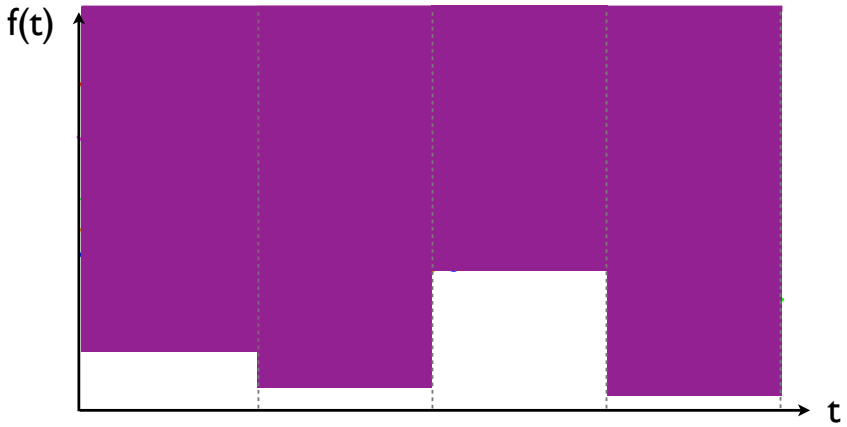


How to approximate $\{f_1, f_2, f_3, f_4\}$?

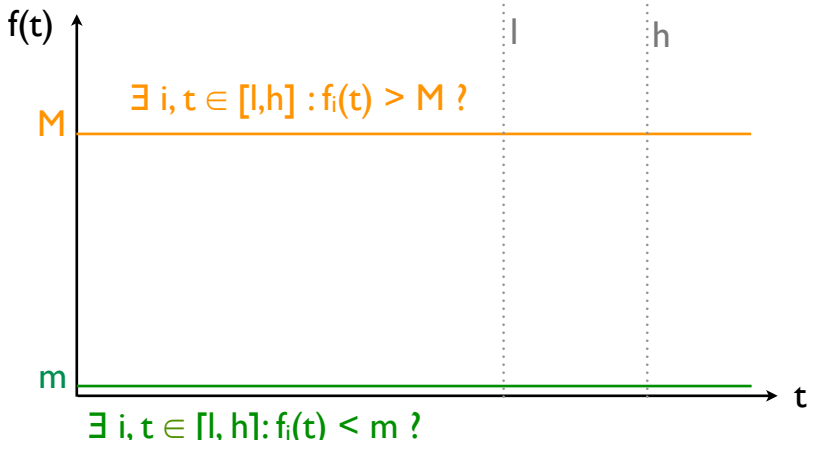
Abstraction of set of functions



A less precise abstraction

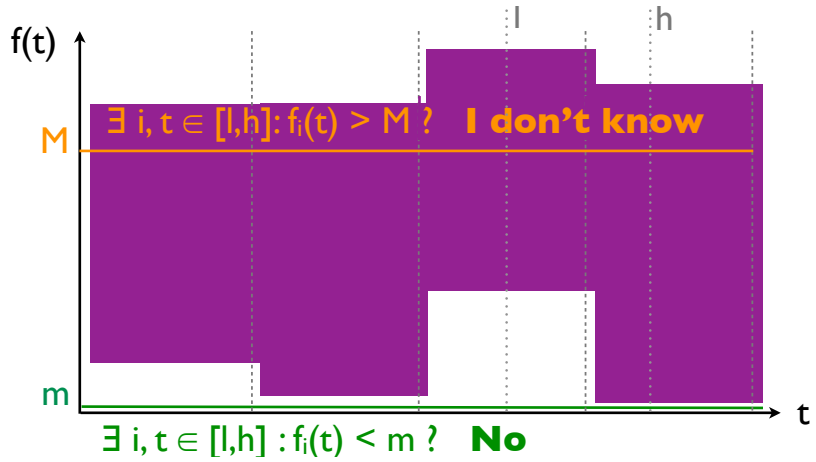


Concrete questions on the f_i



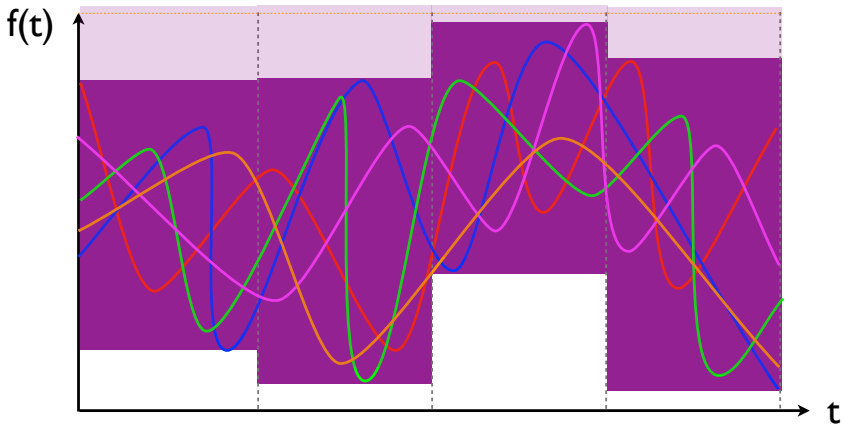
Min/max questions on the f_i

Concrete questions answered in the abstract



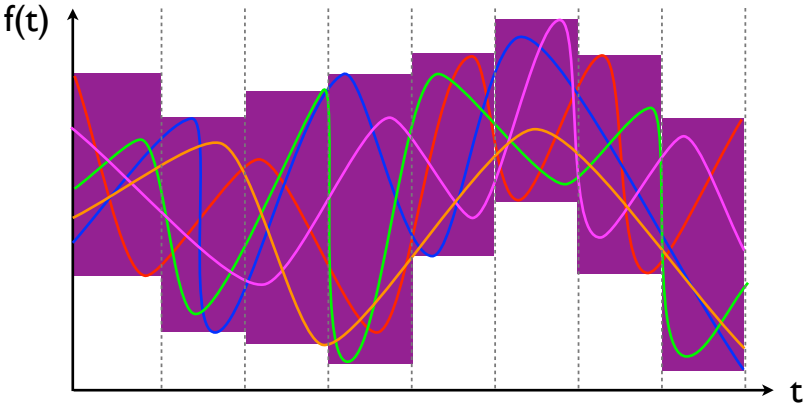
Min/max questions on the f_i

Soundness of the abstraction

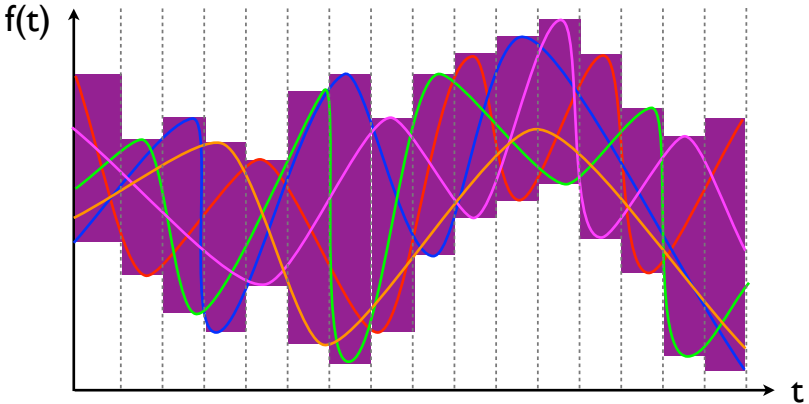


No concrete case is ever forgotten.

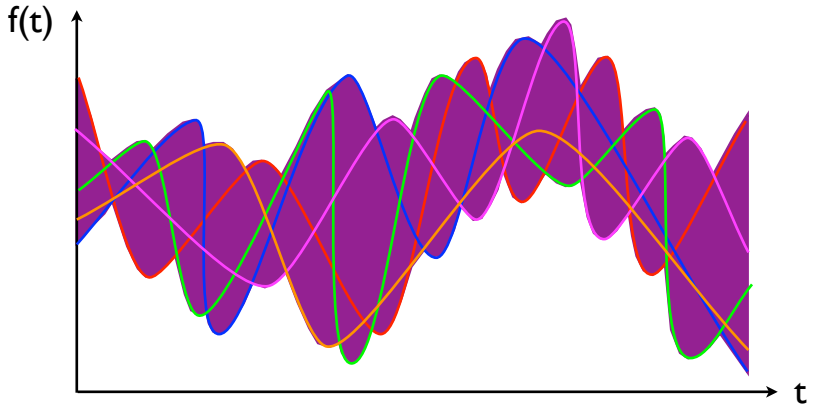
A more precise/refined abstraction



A even more precise/refined abstraction

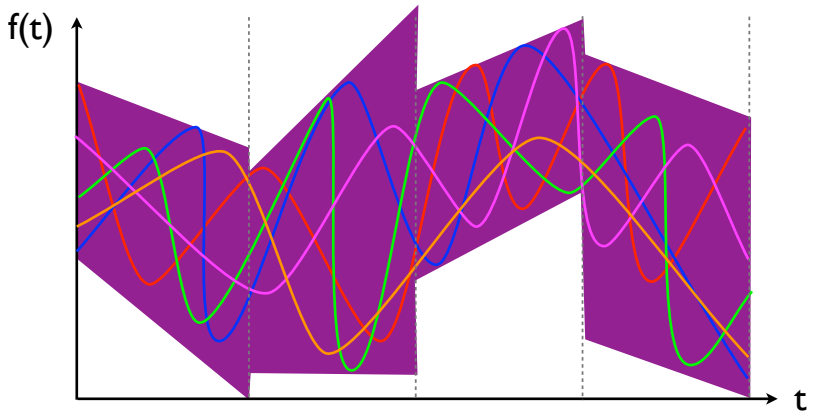


Passing to the limit



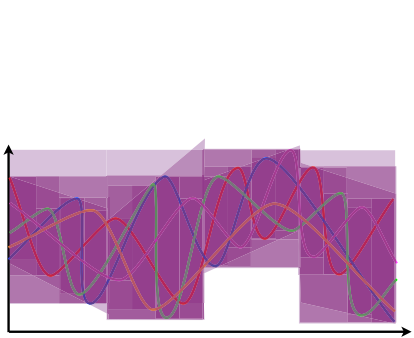
Sound and *complete* abstraction for min/max questions on the f_i

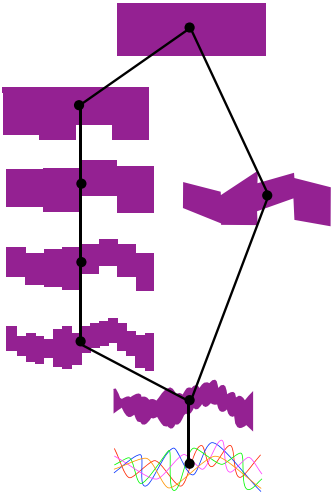
A non-comparable abstraction



Sound and *incomplete* abstraction for min/max questions on the f_i

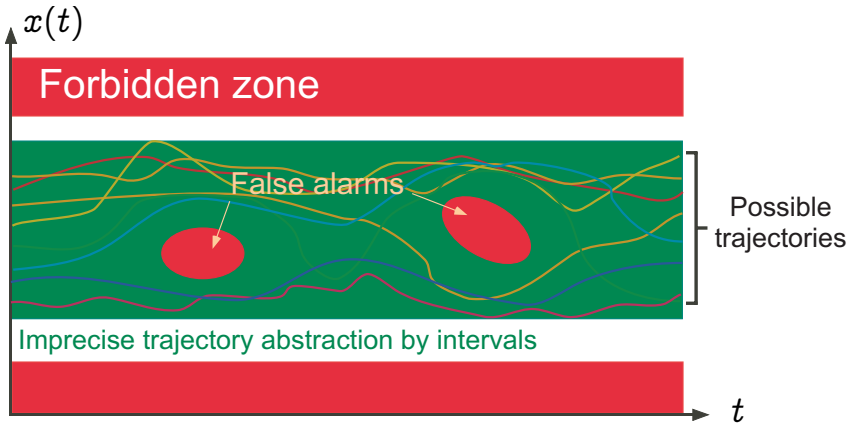
The hierarchy of abstractions





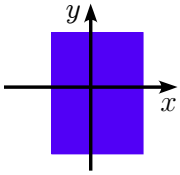
Agenda

False alarms (e.g. interval analysis)



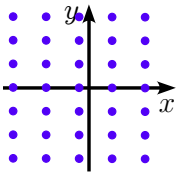
$\Rightarrow$ need for abstraction refinement

Examples of numerical abstract domains



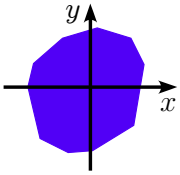
intervals

$x, y \in [a, b]$



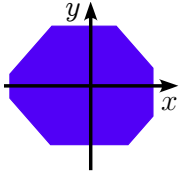
congruences

$x, y \in a\mathbb{Z} + b$



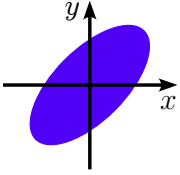
polyhedra

$\bigwedge \sum_i a_i x_i \leq b$



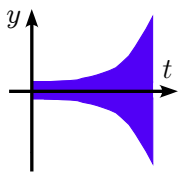
octagons

$\bigwedge \pm x \pm y \leq c$



ellipsoids

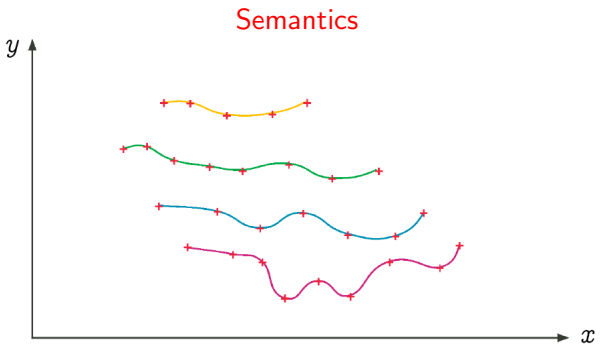
$x^2 + by^2 - axy \leq d$



geometric deviations

$|y| \leq a(1 + b)^{kt}$

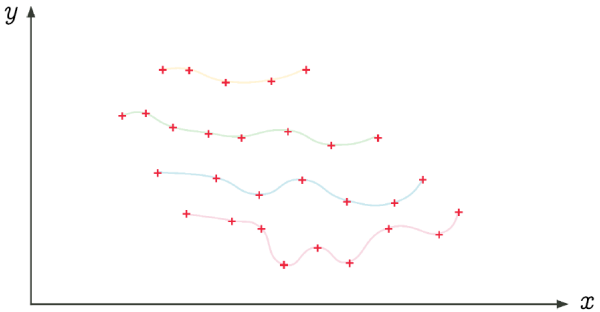
Combinations of abstractions



(In)finite sets of (in)finite traces

Combinations of abstractions

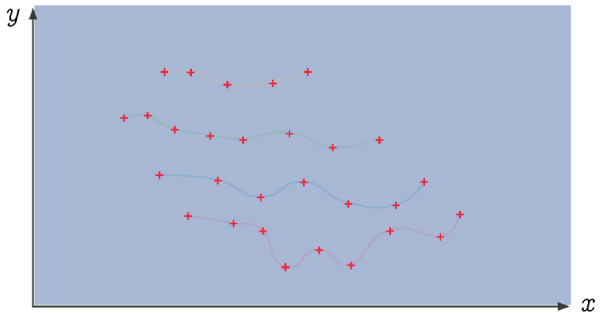
Invariant abstraction: set of reachable states



Set of points (x_i, y_i), Hoare logic

Combinations of abstractions

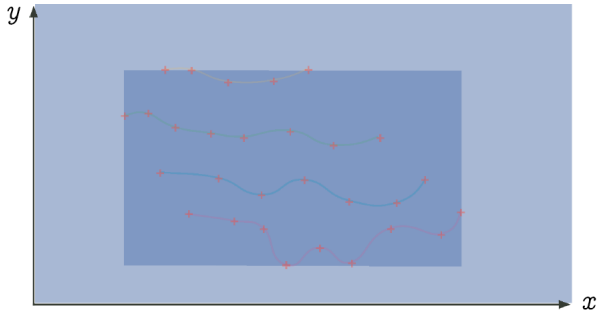
Sign abstraction



$x \geq 0,\ y \geq 0$

Combinations of abstractions

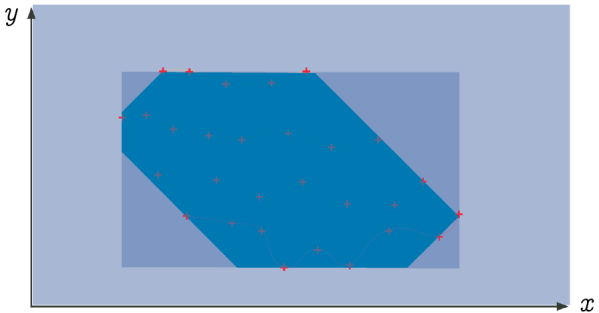
Interval abstraction



$a \leq x \leq b,\ c \leq y \leq d$

Combinations of abstractions

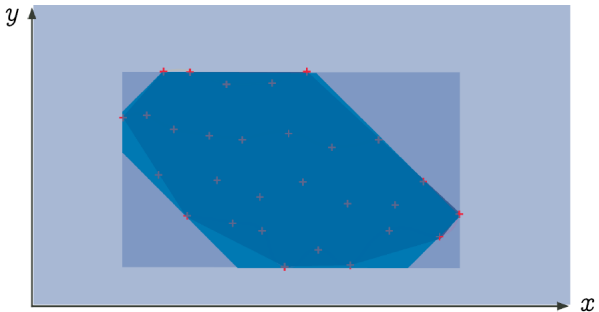
Octagon abstraction



$$\pm x \pm y \leq b$$

Combinations of abstractions

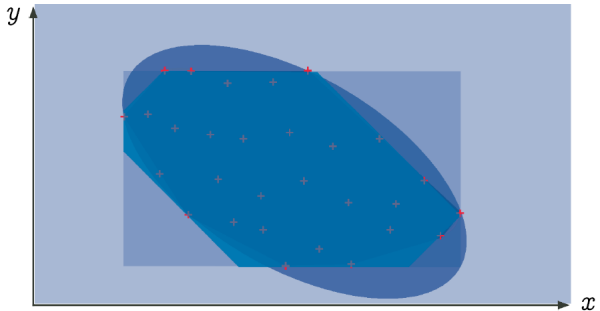
Polyhedral abstraction



$$a \cdot x + b \cdot y \leq c$$

Combinations of abstractions

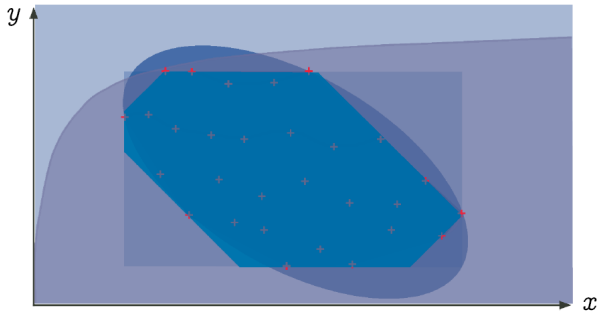
Ellipsoidal abstraction



$$(x-a)^2+(y-b)^2 \leq c$$

Combinations of abstractions

Exponential abstraction



$a^x \leq y$

Agenda

Static analysis of synchronous **C** programs

Concrete semantics and specification

Concrete semantics source

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

Static analysis of synchronous **C** programs

Concrete semantics and specification

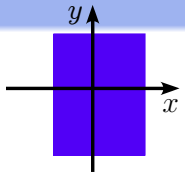
Concrete semantics source

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

Run-time errors

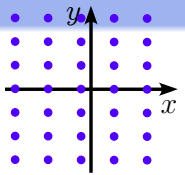
- overflows in float, integer, `enum` arithmetic and cast
- division, modulo by 0 on integers and floats
- invalid pointer arithmetic or dereferencing
- violation of user-specified assertions

Some of ASTRÉE's numerical abstract domains



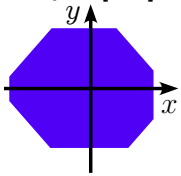
intervals

$x, y \in [a, b]$



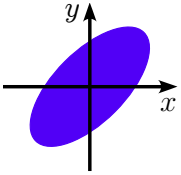
congruences

$x, y \in a\mathbb{Z} + b$



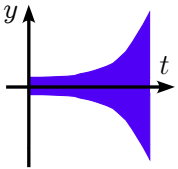
octagons

$\bigwedge \pm x \pm y \leq c$



ellipsoids

$x^2 + by^2 - axy \leq d$



geometric deviations

$|y| \leq a(1 + b)^{kt}$

relational domains are necessary to infer precise bounds

The ASTRÉE static analyser

Analyseur Statique [illegible]

A static analyzer for **C** programs

- developed by CNRS/ENS (from 2002) and AbsInt GmbH
- commercialised by AbsInt since 2010

The ASTRÉE static analyser

Analyseur Statique

A static analyzer for **C** programs

- developed by CNRS/ENS (from 2002) and AbsInt GmbH
- commercialised by AbsInt since 2010

Characteristics

1. sound and automatic, scales up to very large programs
2. specialised for control-command programs ⇒ few false alarms
3. parametric ⇒ fine-tuning by end-users

The ASTRÉE static analyser

Analyseur Statique

A static analyzer for C programs

- developed by CNRS/ENS (from 2002) and AbsInt GmbH
- commercialised by AbsInt since 2010

Characteristics

1. sound and automatic, scales up to very large programs
2. specialised for control-command programs ⇒ few false alarms
3. parametric ⇒ fine-tuning by end-users

Deployment at Airbus

from 2012 A380/A400M/A350 fly-by-wire control software (DAL A)

successful proofs of absence of *run-time error*

≃ 6 hours for 700,000 lines of **C**

from 2016 all new software products (on-going effort)

AIRBUS

Back to Sendmail buffer overflow

()()()...()

email address parsing

true alarms detected

```
#define BUFFERSIZE 200
#define TRUE 1
#define FALSE 0
int copy_it ( char * input , unsigned int length ) {
    char c , localbuf [ BUFFERSIZE ];
    unsigned int upperlimit, quotation, roundquote, inputIndex, outputIndex;
    upperlimit = BUFFERSIZE - 10;
    quotation = roundquote = FALSE ;
    inputIndex = outputIndex = 0;
    while ( inputIndex < length ) {
        c = input [ inputIndex ++];
        if (( c == '<') && (! quotation )) {
            quotation = TRUE ; upperlimit --;
        }
        if (( c == '>') && ( quotation )) {
            quotation = FALSE ; upperlimit ++;
        }
        if (( c == '(') && (! quotation ) && ! roundquote ) {
            roundquote = TRUE ; //upperlimit--; // decrementation was missing
in bug
        }
        if (( c == ')') && (! quotation ) && roundquote ) {
            roundquote = FALSE ; upperlimit ++;
        }
        // If there is sufficient space in the buffer , write the character .
        if ( outputIndex < upperlimit ) {
            localbuf [ outputIndex ] = c;
            outputIndex ++;
        }
    }
    if ( roundquote ) {
        localbuf [ outputIndex ] = ')'; outputIndex ++; }
    if ( quotation ) {
        localbuf [ outputIndex ] = '>'; outputIndex ++; }

    return 0;
```

AIRBUS

Back to Sendmail buffer overflow

fixed

email address parsing

but false alarms still

```
#define BUFFERSIZE 200
#define TRUE 1
#define FALSE 0
int copy_it ( char * input , unsigned int length ) {
    char c , localbuf [ BUFFERSIZE ];
    unsigned int upperlimit, quotation, roundquote, inputIndex, outputIndex;
    upperlimit = BUFFERSIZE - 10;
    quotation = roundquote = FALSE ;
    inputIndex = outputIndex = 0;
    while ( inputIndex < length ) {
        c = input [ inputIndex ++];
        if (( c == '<') && (! quotation )) {
            quotation = TRUE ; upperlimit --;
        }
        if (( c == '>') && ( quotation )) {
            quotation = FALSE ; upperlimit ++;
        }
        if (( c == '(') && (! quotation ) && ! roundquote ) {
            roundquote = TRUE ; upperlimit--; // decrementation was missing in
bug
        }
        if (( c == ')') && (! quotation ) && roundquote ) {
            roundquote = FALSE ; upperlimit ++;
        }
        // If there is sufficient space in the buffer , write the character .
        if ( outputIndex < upperlimit ) {
            localbuf [ outputIndex ] = c;
            outputIndex ++;
        }
    }
    if ( roundquote ) {
        localbuf [ outputIndex ] = ')'; outputIndex ++; }
    if ( quotation ) {
        localbuf [ outputIndex ] = '>'; outputIndex ++; }

    return 0;
```

Back to Sendmail buffer overflow

fixed and tuned

email address parsing

zero alarm

```
#define BUFFERSIZE 200
#define TRUE 1
#define FALSE 0
int copy_it ( char * input , unsigned int length ) {
    char c , localbuf [ BUFFERSIZE ];
    unsigned int upperlimit, quotation, roundquote, inputIndex, outputIndex;
__ASTREE_boolean_pack((upperlimit, inputIndex, outputIndex ; quotation,
roundquote));
    upperlimit = BUFFERSIZE - 10;
    quotation = roundquote = FALSE ;
    inputIndex = outputIndex = 0;
    while ( inputIndex < length ) {
        c = input [ inputIndex ++];
        if (( c == '<') && (! quotation )) {
            quotation = TRUE ; upperlimit --;
        }
        if (( c == '>') && ( quotation )) {
            quotation = FALSE ; upperlimit ++;
        }
        if (( c == '(') && (! quotation ) && ! roundquote ) {
            roundquote = TRUE ; upperlimit--; // decrementation was missing in
bug
        }
        if (( c == ')') && (! quotation ) && roundquote ) {
            roundquote = FALSE ; upperlimit ++;
        }
        // If there is sufficient space in the buffer , write the character .
        if ( outputIndex < upperlimit ) {
            localbuf [ outputIndex ] = c;
            outputIndex ++;
        }
__ASTREE_assert((outputIndex <= BUFFERSIZE-10));
    }
    if ( roundquote ) {
        localbuf [ outputIndex ] = ')'; outputIndex ++; }
    if ( quotation ) {
```

AIRBUS

Static analysis of parallel **C** programs

Concrete semantics and specification

Model: real-time operating system

- fixed set of concurrent threads on a single processor
- shared memory (implicit communications)
- synchronisation primitives (fixed set of mutexes)
- real-time scheduling with fixed priorities (priority-based)

  e.g. A380/A400M/A350 IMA, A350 ASF, SA/LR ATSU
- mono-threaded startup $\neq$ multi-threaded run (restriction)

Static analysis of parallel C programs

Concrete semantics and specification

Model: real-time operating system

- fixed set of concurrent threads on a single processor
- shared memory (implicit communications)
- synchronisation primitives (fixed set of mutexes)
- real-time scheduling with fixed priorities (priority-based)

  e.g. A380/A400M/A350 IMA, A350 ASF, SA/LR ATSU
- mono-threaded startup $\neq$ multi-threaded run (restriction)

Run-time errors

- classic C run-time errors (overflows, invalid pointers, etc.)
- unprotected data-races (report & factor in the analysis)
- incorrect system calls, deadlocks
- but NOT livelocks, priority inversions

AIRBUS

The ASTRÉEA prototype static analyser

An extension of ASTRÉE

- Analyseur Statique de logiciels Temps-RÉel Embarqués Asynchrones
- developed by CNRS/ENS/INRIA since 2009

13/10/2015 1st official merge with AbsInt's commercial ASTRÉE

The ASTRÉEA prototype static analyser

An extension of ASTRÉE

- Analyseur Statique de logiciels Temps-RÉel Embarqués Asynchrones
- developed by CNRS/ENS/INRIA since 2009

13/10/2015 1st official merge with AbsInt's commercial ASTRÉE

results with Airbus

static analysis of A380 FWS

- 15 processes
- $\simeq$ 2 million lines of **C**
- nested loops, complex data structures

$\simeq$ 900 alarms

Schéma producteur(s)-consommateur(s)

partageant un tampon circulaire au moyen de moniteurs *POSIX*

producteur

produire ressource v
lock(m);
while ($i + 1 \equiv_N j$) **do wait**(c, m);
$b[i] \leftarrow v$;
$i \leftarrow i \oplus_N 1$;
signal(c');
unlock(m)

consommateur

lock(m);
while ($i = j$) **do wait**(c', m);
$w \leftarrow b[j]$;
$j \leftarrow j \oplus_N 1$;
unlock(m);
signal(c);
consommer ressource w

Schéma producteur(s)-consommateur(s)

erratum : signal hors section critique

producteur

produire ressource v
lock(m);
while ($i + 1 \equiv_N j$) **do wait**(c, m);
$b[i] \leftarrow v$;
$i \leftarrow i \oplus_N 1$;
signal(c');
unlock(m)

consommateur

lock(m);
while ($i = j$) **do wait**(c', m);
$w \leftarrow b[j]$;
$j \leftarrow j \oplus_N 1$;
unlock(m);
signal(c); WARN: mutex unlocked
consommer ressource w

WARNS: data-races (dans le modèle)

Schéma producteur(s)-consommateur(s)

partageant un tampon circulaire au moyen de moniteurs *POSIX*

producteur

produire ressource v
lock(m);
while ($i + 1 \equiv_N j$) **do wait**(c, m);
$b[i] \leftarrow v$;
$i \leftarrow i \oplus_N 1$;
signal(c');
unlock(m)

consommateur

lock(m);
while ($i = j$) **do wait**(c', m);
$w \leftarrow b[j]$;
$j \leftarrow j \oplus_N 1$;
signal(c);
unlock(m);
consommer ressource w

Schéma producteur(s)-consommateur(s)

erratum: incrémentation simple

producteur

produire ressource v
lock(m);
while ($i + 1 \equiv_N j$) **do wait**(c, m);
$b[i] \leftarrow v$; WARN: array out of bounds
$i \leftarrow i+1$; WARN: integer overflow
signal(c');
unlock(m)

consommateur

lock(m);
while ($i = j$) **do wait**(c', m);
$w \leftarrow b[j]$;
$j \leftarrow j \oplus_N 1$;
signal(c);
unlock(m);
consommer ressource w

Schéma producteur(s)-consommateur(s)

erratum : verrou non pris

producteur	consommateur
produire ressource v	
lock(m);	~~**lock**(m);~~
while ($i + 1 \equiv_N j$) **do wait**(c, m);	**while** ($i = j$) **do wait**(c', m);
	WARN: mutex not owned
$b[i] \leftarrow v$;	$w \leftarrow b[j]$;
$i \leftarrow i \oplus_N 1$;	$j \leftarrow j \oplus_N 1$;
signal(c');	**signal**(c); WARN: mutex unlocked
unlock(m)	~~**unlock**(m);~~
	consommer ressource w

WARNS: data-races

Schéma producteur(s)-consommateur(s)

erratum : mauvais mutex

producteur

produire ressource v
lock(m);
while ($i + 1 \equiv_N j$) **do wait**(c, m);
$b[i] \leftarrow v$;
$i \leftarrow i \oplus_N 1$;
signal(c'); WARN: mutex unlocked
unlock(m)

consommateur

lock(m');
while ($i = j$) **do wait**(c', m');
$w \leftarrow b[j]$;
$j \leftarrow j \oplus_N 1$;
signal(c); WARN: mutex unlocked
unlock(m');
consommer ressource w

WARNS: data-races

Agenda

AIRBUS

Operational semantics

Operational semantics

Mathematical description of the execution of programs

1. a **model** of programs: **transition systems**
   - definition, a **small step semantics**
   - example: a simple imperative language
2. **trace semantics:** a families of **big step** semantics
   - **finite** and **infinite** executions
   - **fixpoint**-based definitions

Agenda

Definition

We will characterize a program by:

- **states**: photography of the program status at an instant of the execution
- **execution steps**: how do we move from one state to the next one

Definition: transition systems (TS)

A **transition system** is a tuple $(\mathbb{S}, \rightarrow)$ where:

- $\mathbb{S}$ is the **set of states** of the system
- $\rightarrow \subseteq \mathcal{P}(\mathbb{S} \times \mathbb{S})$ is the **transition relation** of the system

Note:

- the set of states **may be infinite**

A simple imperative language: syntax

We now look at a more classical **imperative language** (intuitively, a bare-bone subset of C):

- **variables** $\mathbb{X}$: finite, predefined set of variables
- **labels** $\mathbb{L}$: before and after each statement
- **values** $\mathbb{V}$: $\mathbb{V}_{\text{int}} \cup \mathbb{V}_{\text{float}} \cup \dots$

Syntax

e	::=	$v \in \mathbb{V}_{\text{int}} \cup \mathbb{V}_{\text{float}} \cup \dots \mid \text{e} + \text{e} \mid \text{e} * \text{e} \mid \dots$	expressions
c	::=	TRUE $\mid$ FALSE $\mid$ e $<$ e $\mid$ e $=$ e	conditions
i	::=	$x :=$ e;	assignment
	$\mid$	**if**(c) b **else** b	condition
	$\mid$	**while**(c) b	loop
b	::=	{i; ... ; i;}	block, program($\mathbb{P}$)

A simple imperative language: states

A **non-error state** should fully describe the configuration at one instant of the program execution:

- the **memory state** defines the current contents of the memory

$$m \in \mathbb{M} = \mathbb{X} \longrightarrow \mathbb{V}$$

- the **control state** defines *where* the program currently is
  - analoguous to the program counter
  - can be defined by adding **labels** $\mathbb{L} = \{\ell_0, \ell_1, \ldots\}$ between each pair of consecutive statements; then:

$$\mathbb{S} = \mathbb{L} \times \mathbb{M} \uplus \{\Omega\}$$

A simple imperative language: semantics of expressions

- The **semantics $[\![e]\!]$ of expression** e should evaluate each expression into a value, given a memory state
- **Evaluation errors** may occur: division by zero... error value is also noted Ω

Thus: $[\![e]\!] : \mathbb{M} \longrightarrow \mathbb{V} \uplus \{\Omega\}$

Definition, by **induction over the syntax:**

$$
\begin{array}{rcll}
[\![v]\!](m) & = & v & \\
[\![x]\!](m) & = & m(x) & \\
[\![e_0 + e_1]\!](m) & = & [\![e_0]\!](m) \underline{+} [\![e_1]\!](m) & \\
[\![e_0 / e_1]\!](m) & = & \begin{cases} \Omega & \text{if } [\![e_1]\!](m) = 0 \\ [\![e_0]\!](m) \underline{/} [\![e_1]\!](m) & \text{otherwise} \end{cases} &
\end{array}
$$

where $\underline{\oplus}$ is the machine implementation of operator $\oplus$, and is Ω-strict, i.e., $\forall v \in \mathbb{V},\ v \underline{\oplus} \Omega = \Omega \underline{\oplus} v = \Omega$.

A simple imperative language: semantics of conditions

- The **semantics $[\![c]\!]$ of condition** c should return a *boolean value*
- It follows a similar definition to that of the semantics of expressions:
  $[\![c]\!] : \mathbb{M} \longrightarrow \mathbb{V}_{\text{bool}} \uplus \{\Omega\}$

Definition, by **induction over the syntax**:

$$
\begin{aligned}
[\![\texttt{TRUE}]\!](m) &= \texttt{TRUE} \\
[\![\texttt{FALSE}]\!](m) &= \texttt{FALSE} \\
[\![e_0 < e_1]\!](m) &= \begin{cases} \texttt{TRUE} & \text{if } [\![e_0]\!](m) < [\![e_1]\!](m) \\ \texttt{FALSE} & \text{if } [\![e_0]\!](m) \geq [\![e_1]\!](m) \\ \Omega & \text{if } [\![e_0]\!](m) = \Omega \text{ or } [\![e_1]\!](m) = \Omega \end{cases} \\
[\![e_0 = e_1]\!](m) &= \begin{cases} \texttt{TRUE} & \text{if } [\![e_0]\!](m) = [\![e_1]\!](m) \\ \texttt{FALSE} & \text{if } [\![e_0]\!](m) \neq [\![e_1]\!](m) \\ \Omega & \text{if } [\![e_0]\!](m) = \Omega \text{ or } [\![e_1]\!](m) = \Omega \end{cases}
\end{aligned}
$$

A simple imperative language: transitions

We now consider the transition induced by each statement.

Case of **assignment** $\ell_0 : x = \mathtt{e};\ \ell_1$

- if $[\![\mathtt{e}]\!](m) \neq \Omega$, then $(\ell_0, m) \rightarrow (\ell_1, m[x \leftarrow [\![\mathtt{e}]\!](m)])$
- if $[\![\mathtt{e}]\!](m) = \Omega$, then $(\ell_0, m) \rightarrow \Omega$

Case of **condition** $\ell_0 : \mathbf{if}(\mathtt{c})\{\ell_1 : \mathtt{b}_t\ \ell_2\}\ \mathbf{else}\{\ell_3 : \mathtt{b}_f\ \ell_4\}\ \ell_5$

- if $[\![\mathtt{c}]\!](m) = \mathtt{TRUE}$, then $(\ell_0, m) \rightarrow (\ell_1, m)$
- if $[\![\mathtt{c}]\!](m) = \mathtt{FALSE}$, then $(\ell_0, m) \rightarrow (\ell_3, m)$
- if $[\![\mathtt{c}]\!](m) = \Omega$, then $(\ell_0, m) \rightarrow \Omega$
- $(\ell_2, m) \rightarrow (\ell_5, m)$
- $(\ell_4, m) \rightarrow (\ell_5, m)$

A simple imperative language: transitions

Case of **loop** $\ell_0 : \textbf{while}(\mathtt{c})\{\ell_1 : \mathtt{b}_t\ \ell_2\}\ \ell_3$

- if $[\![\mathtt{c}]\!](m) = \mathtt{TRUE}$, then $\begin{cases} (\ell_0, m) \to (\ell_1, m) \\ (\ell_2, m) \to (\ell_1, m) \end{cases}$
- if $[\![\mathtt{c}]\!](m) = \mathtt{FALSE}$, then $\begin{cases} (\ell_0, m) \to (\ell_3, m) \\ (\ell_2, m) \to (\ell_3, m) \end{cases}$
- if $[\![\mathtt{c}]\!](m) = \Omega$, then $\begin{cases} (\ell_0, m) \to \Omega \\ (\ell_2, m) \to \Omega \end{cases}$

Case of $\{\ell_0 : \mathtt{i}_0; \ell_1 : \ldots; \ell_{n-1}\mathtt{i}_{n-1}; \ell_n\}$

- the transition relation is defined by the individual instructions

Extending the language with non-determinism

The language we have considered so far is a bit **limited**:

- it is **deterministic**: at most one transition possible from any state
- it does not support the **input of values**

Changes if we model non deterministic inputs...

... with an input instruction:

- $\texttt{i} ::= \ldots \mid x := \textbf{input}()$
- $\ell_0 : x := \textbf{input}(); \ell_1$ generates transitions

$$\forall v \in \mathbb{V},\ (\ell_0, m) \rightarrow (\ell_1, m[x \leftarrow v])$$

- one instruction induces non determinism

Semantics of real world programming languages

C language:

- several **norms**: ANSI C'99, ANSI C'11, K&R...
- not fully specified:
  - **undefined behavior**
  - **implementation dependent behavior**: architecture (ABI) or implementation (compiler...)
  - unspecified parts: leave room for implementation of compilers and optimizations
- **formalizations** in HOL (C'99), in Coq (CompCert C compiler)

OCaml language:

- more formal...
- ... but still with some unspecified parts, e.g., execution order

Agenda

AIRBUS

Execution traces

- So far, we considered only states and atomic transitions
- We now consider program **executions** as a whole

Definition: traces

- A **finite trace** is a finite sequence of states $s_0, \ldots, s_n$, noted $\langle s_0, \ldots, s_n \rangle$
- An **infinite trace** is an infinite sequence of states $\langle s_0, \ldots \rangle$

Besides, we write:

- $\mathbb{S}^\star$ for the **set of finite traces**
- $\mathbb{S}^\omega$ for the **set of infinite traces**
- $\mathbb{S}^\infty = \mathbb{S}^\star \cup \mathbb{S}^\omega$ for the **set of finite or infinite traces**

Semantics of finite traces

We consider a transition system $\mathcal{S} = (\mathbb{S}, \rightarrow)$

Definition

The **finite traces semantics** $[\![\mathcal{S}]\!]^\star$ is defined by:

$$[\![\mathcal{S}]\!]^\star = \{\langle s_0, \ldots, s_n \rangle \in \mathbb{S}^\star \mid \forall i,\ s_i \rightarrow s_{i+1}\}$$

Example:

- contrived transition system $\mathcal{S} = (\{a, b, c, d\}, \{(a, b), (b, a), (b, c)\})$
- finite traces semantics:

$$\begin{array}{rcll} [\![\mathcal{S}]\!]^\star & = & \{ & \langle a, b, \ldots, a, b, a \rangle, \quad \langle b, a, \ldots, a, b, a \rangle, \\ & & & \langle a, b, \ldots, a, b, a, b \rangle, \quad \langle b, a, \ldots, a, b, a, b \rangle, \\ & & & \langle a, b, \ldots, a, b, a, b, c \rangle, \quad \langle b, a, \ldots, a, b, a, b, c \rangle \\ & & & \langle c \rangle, \quad \langle d \rangle \quad \} \end{array}$$

Example: imperative program

Similarly, we can write the traces of a simple imperative program:

$$
\begin{array}{ll}
\ell_0: & \mathtt{x} := 1; \\
\ell_1: & \mathtt{y} := 0; \\
\ell_2: & \textbf{while}(\mathtt{x} < 4)\{ \\
\ell_3: & \qquad \mathtt{y} := \mathtt{y} + \mathtt{x}; \\
\ell_4: & \qquad \mathtt{x} := \mathtt{x} + 1; \\
\ell_5: & \} \\
\ell_6: & \text{(final program point)}
\end{array}
$$

$$
\begin{aligned}
\tau \quad = \quad \langle \quad & (\ell_0, (\!|\mathtt{x} = \mathtt{x}_0, \mathtt{y} = \mathtt{y}_0|\!)), (\ell_1, (\!|\mathtt{x} = 1, \mathtt{y} = \mathtt{y}_0|\!)), \\
& (\ell_2, (\!|\mathtt{x} = 1, \mathtt{y} = 0|\!)), (\ell_3, (\!|\mathtt{x} = 1, \mathtt{y} = 0|\!)), \\
& (\ell_4, (\!|\mathtt{x} = 1, \mathtt{y} = 1|\!)), (\ell_5, (\!|\mathtt{x} = 2, \mathtt{y} = 1|\!)), \\
& (\ell_3, (\!|\mathtt{x} = 2, \mathtt{y} = 1|\!)), (\ell_4, (\!|\mathtt{x} = 2, \mathtt{y} = 3|\!)), \\
& (\ell_5, (\!|\mathtt{x} = 3, \mathtt{y} = 3|\!)), (\ell_3, (\!|\mathtt{x} = 3, \mathtt{y} = 3|\!)), \\
& (\ell_4, (\!|\mathtt{x} = 3, \mathtt{y} = 6|\!)), (\ell_5, (\!|\mathtt{x} = 4, \mathtt{y} = 6|\!)), \\
& (\ell_6, (\!|\mathtt{x} = 4, \mathtt{y} = 6|\!)) \quad \rangle
\end{aligned}
$$

- very **precise** description of what the program does...
- ... but **quite cumbersome**

Trace semantics fixpoint form

We define a **semantic function**, that computes the traces of length $i+1$ from the traces of length i (where $i \geq 1$):

Finite traces semantics as a fixpoint

Let $\mathcal{I} = \{\epsilon\} \uplus \{\langle s \rangle \mid s \in \mathbb{S}\}$.
Let $F_\star$ be the function defined by:

$$\begin{array}{rccl} F_\star : & \mathcal{P}(\mathbb{S}^\star) & \longrightarrow & \mathcal{P}(\mathbb{S}^\star) \\ & X & \longmapsto & X \cup \{\langle s_0, \ldots, s_n, s_{n+1} \rangle \mid \langle s_0, \ldots, s_n \rangle \in X \wedge s_n \to s_{n+1}\} \end{array}$$

Then, $F_\star$ is **continuous** and thus has a least-fixpoint greater than $\mathcal{I}$; moreover:

$$\mathsf{lfp}_{\mathcal{I}} F_\star = [\![\mathcal{S}]\!]^\star = \bigcup_{n \in \mathbb{N}} F_\star^n(\mathcal{I})$$

Trace semantics fixpoint form: example

Example, with the same simple transition system $\mathcal{S} = (\mathbb{S}, \rightarrow)$:

- $\mathbb{S} = \{a, b, c, d\}$
- $\rightarrow$ is defined by $a \rightarrow b$, $b \rightarrow a$ and $b \rightarrow c$

Then, the first iterates are:

$$
\begin{array}{rcl}
F^0_\star(\mathcal{I}) & = & \{\epsilon, \langle a\rangle, \langle b\rangle, \langle c\rangle, \langle d\rangle\} \\
F^1_\star(\mathcal{I}) & = & F^0_\star(\mathcal{I}) \cup \{\langle b, a\rangle, \langle a, b\rangle, \langle b, c\rangle\} \\
F^2_\star(\mathcal{I}) & = & F^1_\star(\mathcal{I}) \cup \{\langle a, b, a\rangle, \langle b, a, b\rangle, \langle a, b, c\rangle\} \\
F^3_\star(\mathcal{I}) & = & F^2_\star(\mathcal{I}) \cup \{\langle b, a, b, a\rangle, \langle a, b, a, b\rangle, \langle b, a, b, c\rangle\} \\
F^4_\star(\mathcal{I}) & = & F^3_\star(\mathcal{I}) \cup \{\langle a, b, a, b, a\rangle, \langle b, a, b, a, b\rangle, \langle a, b, a, b, c\rangle\} \\
F^5_\star(\mathcal{I}) & = & \ldots
\end{array}
$$

- the traces of $[\![\mathcal{S}]\!]^\star$ of length $n+1$ appear in $F^n_\star(\mathcal{I})$

Agenda

Summary

We have discussed:

- **small-step / structural operational semantics:** individual program steps
- **big-step / natural semantics:** program executions as sequences of transitions
- their **fixpoint definitions** and properties

Next:

- another family of semantics, **more compact** and **compositional**
- **semantic program** and **proof methods**

Agenda

Introduction

Operational semantics

Defined as small execution steps *(transition relation)*
over low-level internal configurations *(states)*

Transitions are chained to define *(maximal)* traces
possibly abstracted as input-output relations *(big-step)*

Denotational semantics

Direct functions from programs to mathematical objects *(denotations)*
by induction on the program syntax *(compositional)*
ignoring intermediate steps and execution details *(no state)*

$\Longrightarrow$ Higher-level, more abstract, more modular.
Tries to decouple a program meaning from its execution.
Focus on the mathematical structures that represent programs.
(founded by Strachey and Scott in the 70s: [Scott-Strachey71])

"Assembly" of semantics vs. "Functional programming" of semantics

Denotation worlds

- **imperative programs**

  effect of a program: mutate a memory state
  natural denotation: input/output function
  $\mathcal{D} \simeq \mathit{memory} \to \mathit{memory}$

  challenge: build a whole program denotation
  from denotations of atomic language constructs (modularity)

$\Longrightarrow$ very rich theory of mathematical structures
(Scott domains, cartesian closed categories, coherent spaces, event structures, game semantics, etc. We will not present them in this overview!)

Agenda

A simple imperative language: IMP

IMP expressions

$$
\begin{array}{rcll}
\mathit{expr} & ::= & X & (\textit{variable}) \\
 & | & c & (\textit{constant}) \\
 & | & \diamond\, \mathit{expr} & (\textit{unary operation}) \\
 & | & \mathit{expr} \diamond \mathit{expr} & (\textit{binary operation})
\end{array}
$$

- variables in a fixed set $X \in \mathcal{V}$
- constants $\mathcal{I} \stackrel{\text{def}}{=} \mathbb{B} \cup \mathbb{Z}$:
  - booleans $\mathbb{B} \stackrel{\text{def}}{=} \{\,\text{true}, \text{false}\,\}$
  - integers $\mathbb{Z}$
- operations $\diamond$:
  - integer operations: $+$, $-$, $\times$, $/$, $<$, $\leq$
  - boolean operations: $\neg$, $\wedge$, $\vee$
  - polymorphic operations: $=$, $\neq$

A simple imperative language: IMP

Statements

stat	::=	**skip**	(*do nothing*)
	|	$X \leftarrow expr$	(*assignment*)
	|	*stat*; *stat*	(*sequence*)
	|	**if** *expr* **then** *stat* **else** *stat*	(*conditional*)
	|	**while** *expr* **do** *stat*	(*loop*)

(inspired from the presentation in [Benton96])

Expression semantics

$\mathbb{E}[\![\, expr \,]\!] : \mathcal{E} \rightharpoonup \mathcal{I}$

- environments $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{V} \to \mathcal{I}$ map variables in $\mathcal{V}$ to values in $\mathcal{I}$
- $\mathbb{E}[\![\, expr \,]\!]$ returns a value in $\mathcal{I}$
- $\rightharpoonup$ denotes partial functions (as opposed to $\to$)
  necessary because some operations are undefined
  - $1 + \text{true}$, $1 \wedge 2$ (type mismatch)
  - $3/0$ (invalid value)
- defined by structural induction on abstract syntax trees
  (next slide)

(when we use the notation $\mathbb{X}[\![\, y \,]\!]$, y is a syntactic object; X serves to distinguish between different semantic functions with different signatures, often varying with the kind of syntactic object y (expression, statement, etc.);
$\mathbb{X}[\![\, y \,]\!]z$ is the application of the function $\mathbb{X}[\![\, y \,]\!]$ to the object z)

Expression semantics

$$\underline{\mathbb{E}[\![\, expr \,]\!] : \mathcal{E} \rightharpoonup \mathcal{I}}$$

$$
\begin{array}{lcll}
\mathbb{E}[\![\, c \,]\!]\rho & \stackrel{\text{def}}{=} & c & \in \mathcal{I} \\
\mathbb{E}[\![\, V \,]\!]\rho & \stackrel{\text{def}}{=} & \rho(V) & \in \mathcal{I} \\
\mathbb{E}[\![\, -e \,]\!]\rho & \stackrel{\text{def}}{=} & -v & \in \mathbb{Z} \quad \text{if } v = \mathbb{E}[\![\, e \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, \neg e \,]\!]\rho & \stackrel{\text{def}}{=} & \neg v & \in \mathbb{B} \quad \text{if } v = \mathbb{E}[\![\, e \,]\!]\rho \in \mathbb{B} \\
\mathbb{E}[\![\, e_1 + e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 + v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, e_1 - e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 - v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, e_1 \times e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 \times v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, e_1 / e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 / v_2 & \in \mathbb{Z} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \setminus \{0\} \\
\mathbb{E}[\![\, e_1 \wedge e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 \wedge v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{B}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{B} \\
\mathbb{E}[\![\, e_1 \vee e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 \vee v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{B}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{B} \\
\mathbb{E}[\![\, e_1 < e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 < v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, e_1 \leq e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 \leq v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathbb{Z}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathbb{Z} \\
\mathbb{E}[\![\, e_1 = e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 = v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathcal{I}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathcal{I} \\
\mathbb{E}[\![\, e_1 \neq e_2 \,]\!]\rho & \stackrel{\text{def}}{=} & v_1 \neq v_2 & \in \mathbb{B} \quad \text{if } v_1 = \mathbb{E}[\![\, e_1 \,]\!]\rho \in \mathcal{I}, v_2 = \mathbb{E}[\![\, e_2 \,]\!]\rho \in \mathcal{I}
\end{array}
$$

undefined otherwise

Statement semantics

$\mathbb{S}[\![\, stat \,]\!] : \mathcal{E} \rightharpoonup \mathcal{E}$

- maps an environment before the statement to an environment after the statement
- partial function due to
  - errors in expressions
  - non-termination
- also defined by structural induction

Summary

Rewriting the semantics using total functions on cpos:

- $\mathbb{E}[\![\, expr \,]\!] : \mathcal{E}_\perp \xrightarrow{c} \mathcal{I}_\perp$

  returns $\perp$ for an error or if its argument is $\perp$
- $\mathbb{S}[\![\, stat \,]\!] : \mathcal{E}_\perp \xrightarrow{c} \mathcal{E}_\perp$
  - $\mathbb{S}[\![\, \textbf{skip} \,]\!]\rho \stackrel{\text{def}}{=} \rho$
  - $\mathbb{S}[\![\, e_1; e_2 \,]\!] \stackrel{\text{def}}{=} \mathbb{S}[\![\, e_2 \,]\!] \circ \mathbb{S}[\![\, e_1 \,]\!]$
  - $\mathbb{S}[\![\, X \leftarrow e \,]\!]\rho \stackrel{\text{def}}{=} \begin{cases} \perp & \text{if } \mathbb{E}[\![\, e \,]\!]\rho = \perp \\ \rho[X \mapsto \mathbb{E}[\![\, e \,]\!]\rho] & \text{otherwise} \end{cases}$
  - $\mathbb{S}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!]\rho \stackrel{\text{def}}{=} \begin{cases} \mathbb{S}[\![\, s_1 \,]\!]\rho & \text{if } \mathbb{E}[\![\, e \,]\!]\rho = \text{true} \\ \mathbb{S}[\![\, s_2 \,]\!]\rho & \text{if } \mathbb{E}[\![\, e \,]\!]\rho = \text{false} \\ \perp & \text{otherwise} \end{cases}$
  - $\mathbb{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!] \stackrel{\text{def}}{=} \text{lfp } F$

    where $F(f)(\rho) = \begin{cases} \rho & \text{if } \mathbb{E}[\![\, e \,]\!]\rho = \text{false} \\ f(\mathbb{S}[\![\, s \,]\!]\rho) & \text{if } \mathbb{E}[\![\, e \,]\!]\rho = \text{true} \\ \perp & \text{otherwise} \end{cases}$

Statement semantics: loops

How do we handle loops?

the semantics of loops must satisfy:

$$\mathbb{S}[\![\,\textbf{while}\ e\ \textbf{do}\ s\,]\!]\rho = \begin{cases} \rho & \text{if } \mathbb{E}[\![\,e\,]\!]\rho = \text{false} \\ \mathbb{S}[\![\,\textbf{while}\ e\ \textbf{do}\ s\,]\!](\mathbb{S}[\![\,s\,]\!]\rho) & \text{if } \mathbb{E}[\![\,e\,]\!]\rho = \text{true} \\ \text{undefined} & \text{otherwise} \end{cases}$$

this is a recursive definition, we must prove that:

- the equation has solutions
- choose the right one

$\Longrightarrow$ we use fixpoints on partially ordered sets

Agenda

AIRBUS

Motivation

Are the operational and denotational semantics consistent with each other?

Note that:

- systems are actually described operationally
- the denotational semantics is a more abstract representation
  (more suitable for some reasoning on the system)

$\Longrightarrow$ the denotational semantics must be proven faithful (in some sense) to the operational model to be of any use

Reminder: from traces to big-step semantics

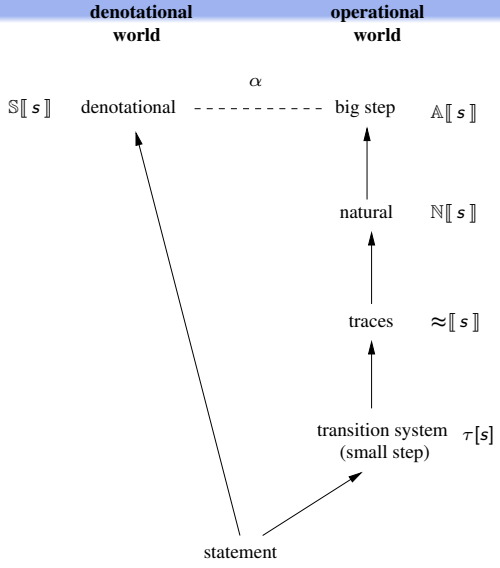
Big-step semantics: abstraction of traces
only remembers the input-output relations

many variants exist:

- "angelic" semantics, in $\mathcal{P}(\Sigma \times \Sigma)$:
  $\mathbb{A}[\![s]\!] \stackrel{\text{def}}{=} \{ (\sigma, \sigma') \mid \exists (\sigma_0, \ldots, \sigma_n) \in t[\![s]\!]^* : \sigma = \sigma_0, \sigma' = \sigma_n \}$
  (only give information on the terminating behaviors;
  can only prove partial correctness)
- natural semantics, in $\mathcal{P}(\Sigma \times \Sigma_\bot)$:
  $\mathbb{N}[\![s]\!] \stackrel{\text{def}}{=} \mathbb{A}[\![s]\!] \cup \{ (\sigma, \bot) \mid \exists (\sigma_0, \ldots) \in t[\![s]\!]^\omega : \sigma = \sigma_0 \}$
  (models the terminating and non-terminating behaviors;
  can prove total correctness)

Semantic diagram

(α is an isomorphism)



Agenda

AIRBUS

Language

Expressions and conditions

$$
\begin{array}{lcll}
\textit{expr} & ::= & V & V \in \mathcal{V} \\
 & | & c & c \in \mathbb{Z} \\
 & | & -\textit{expr} & \\
 & | & \textit{expr} \diamond \textit{expr} & \diamond \in \{+, -, \times, /\} \\
 & | & \textbf{rand}(a, b) & a, b \in \mathbb{Z} \\
\textit{cond} & ::= & \textit{expr} \bowtie \textit{expr} & \bowtie \in \{\leq, \geq, =, \neq, <, >\} \\
 & | & \neg \textit{cond} & \\
 & | & \textit{cond} \diamond \textit{cond} & \diamond \in \{\wedge, \vee\}
\end{array}
$$

Statements

$$
\begin{array}{lcl}
\textit{stat} & ::= & V \leftarrow \textit{expr} \\
 & | & \textbf{if } \textit{cond} \textbf{ then } \textit{stat} \textbf{ else } \textit{stat} \\
 & | & \textbf{while } \textit{cond} \textbf{ do } \textit{stat} \\
 & | & \textit{stat}; \textit{stat} \\
 & | & \textbf{skip}
\end{array}
$$

Concrete semantics

Classic non-deterministic concrete semantics, in denotational style:

$\mathbb{E}[\![expr]\!] : \mathcal{E} \to \mathcal{P}(\mathbb{Z})$ *(arithmetic expressions)*

$$
\begin{array}{ll}
\mathbb{E}[\![V]\!]\rho & \stackrel{\text{def}}{=} \{\rho(V)\} \\
\mathbb{E}[\![c]\!]\rho & \stackrel{\text{def}}{=} \{c\} \\
\mathbb{E}[\![\textbf{rand}(a,b)]\!]\rho & \stackrel{\text{def}}{=} \{\, x \mid a \leq x \leq b \,\} \\
\mathbb{E}[\![-e]\!]\rho & \stackrel{\text{def}}{=} \{\, -v \mid v \in \mathbb{E}[\![e]\!]\rho \,\} \\
\mathbb{E}[\![e_1 \diamond e_2]\!]\rho & \stackrel{\text{def}}{=} \{\, v_1 \diamond v_2 \mid v_1 \in \mathbb{E}[\![e_1]\!]\rho,\ v_2 \in \mathbb{E}[\![e_2]\!]\rho,\ \diamond \neq / \vee v_2 \neq 0 \,\}
\end{array}
$$

$\mathbb{C}[\![cond]\!] : \mathcal{E} \to \mathcal{P}(\{\text{true}, \text{false}\})$ *(boolean conditions)*

$$
\begin{array}{ll}
\mathbb{C}[\![\neg c]\!]\rho & \stackrel{\text{def}}{=} \{\, \neg v \mid v \in \mathbb{C}[\![c]\!]\rho \,\} \\
\mathbb{C}[\![c_1 \diamond c_2]\!]\rho & \stackrel{\text{def}}{=} \{\, v_1 \diamond v_2 \mid v_1 \in \mathbb{C}[\![c_1]\!]\rho,\ v_2 \in \mathbb{C}[\![c_2]\!]\rho \,\} \\
\mathbb{C}[\![e_1 \bowtie e_2]\!]\rho & \stackrel{\text{def}}{=} \{\, \text{true} \mid \exists v_1 \in \mathbb{E}[\![e_1]\!]\rho,\ v_2 \in \mathbb{E}[\![e_2]\!]\rho\text{: } v_1 \bowtie v_2 \,\} \cup \\
 & \quad \{\, \text{false} \mid \exists v_1 \in \mathbb{E}[\![e_1]\!]\rho,\ v_2 \in \mathbb{E}[\![e_2]\!]\rho\text{: } v_1 \not\bowtie v_2 \,\}
\end{array}
$$

where $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{V} \to \mathbb{Z}$

Concrete semantics

$\mathbb{S}[\![\, stat \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$

$$
\begin{array}{ll}
\mathbb{S}[\![\, \textbf{skip} \,]\!] R & \stackrel{\text{def}}{=} R \\
\mathbb{S}[\![\, s_1;\ s_2 \,]\!] R & \stackrel{\text{def}}{=} \mathbb{S}[\![\, s_2 \,]\!](\mathbb{S}[\![\, s_1 \,]\!] R) \\
\mathbb{S}[\![\, V \leftarrow e \,]\!] R & \stackrel{\text{def}}{=} \{\, \rho[V \mapsto v] \mid \rho \in R,\ v \in \mathbb{E}[\![\, e \,]\!]\rho \,\} \\
\mathbb{S}[\![\, \textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2 \,]\!] R & \stackrel{\text{def}}{=} \mathbb{S}[\![\, s_1 \,]\!](\mathbb{S}[\![\, c? \,]\!] R) \cup \mathbb{S}[\![\, s_2 \,]\!](\mathbb{S}[\![\, \neg c? \,]\!] R) \\
\mathbb{S}[\![\, \textbf{while } c \textbf{ do } s \,]\!] R & \stackrel{\text{def}}{=} \mathbb{S}[\![\, \neg c? \,]\!](\text{lfp } \lambda I.\ R \cup \mathbb{S}[\![\, s \,]\!](\mathbb{S}[\![\, c? \,]\!] I))
\end{array}
$$

where

$$
\mathbb{S}[\![\, c? \,]\!] R \stackrel{\text{def}}{=} \{\, \rho \in R \mid \text{true} \in \mathbb{C}[\![\, c \,]\!]\rho \,\}
$$

$\mathbb{S}[\![\, stat \,]\!]$ is a $\cup$−morphism in the complete lattice $(\mathcal{P}(\mathcal{E}), \subseteq, \cup, \cap, \emptyset, \mathcal{E})$

Reminder: Non-relational abstractions

Reminder: we compose two abstractions:

- $\mathcal{P}(\mathcal{V} \to \mathbb{Z})$ is abstracted as $\mathcal{V} \to \mathcal{P}(\mathbb{Z})$ (forget relationship)
- $\mathcal{P}(\mathbb{Z})$ is abstracted as intervals $\mathcal{I}$ (keep only bounds)

Interval expression evaluation

$\mathbb{E}^\sharp[\![\, expr \,]\!] : \mathcal{E}^\sharp \to \mathcal{I}$
interval version of $\mathbb{E}[\![\, expr \,]\!] : \mathcal{E} \to \mathcal{P}(\mathbb{Z})$

Definition by structural induction, very similar to $\mathbb{E}[\![\, expr \,]\!]$

$$
\begin{array}{ll}
\mathbb{E}^\sharp[\![\, V \,]\!]X^\sharp & \stackrel{\text{def}}{=} X^\sharp(V) \\
\mathbb{E}^\sharp[\![\, c \,]\!]X^\sharp & \stackrel{\text{def}}{=} [c, c] \\
\mathbb{E}^\sharp[\![\, \mathbf{rand}(a, b) \,]\!]X^\sharp & \stackrel{\text{def}}{=} [a, b] \\
\mathbb{E}^\sharp[\![\, -e \,]\!]X^\sharp & \stackrel{\text{def}}{=} -^\sharp\ \mathbb{E}^\sharp[\![\, e \,]\!]X^\sharp \\
\mathbb{E}^\sharp[\![\, e_1 \diamond e_2 \,]\!]X^\sharp & \stackrel{\text{def}}{=} \mathbb{E}^\sharp[\![\, e_1 \,]\!]X^\sharp\ \diamond^\sharp\ \mathbb{E}^\sharp[\![\, e_2 \,]\!]X^\sharp
\end{array}
$$

Interval arithmetic

- $-^\sharp\ [a,b] = [-b,-a]$
- $[a,b]\ +^\sharp\ [c,d] = [a+c, b+d]$
- $[a,b]\ -^\sharp\ [c,d] = [a-d, b-c]$
- $\forall i \in \mathcal{I}\text{: } -^\sharp\ \bot = \bot\ +^\sharp\ i = i\ +^\sharp\ \bot = \cdots = \bot$ (strictness)

where: $+$ and $-$ is extended to $+\infty$, $-\infty$ as:
$\forall x \in \mathbb{Z}\text{: } (+\infty) + x = +\infty,\ (-\infty) + x = -\infty,\ -(+\infty) = (-\infty), \ldots$

- $[a,b]\ \times^\sharp\ [c,d] = [\ \min(a\times c,\ a\times d,\ b\times c,\ b\times d),$
  $\max(a\times c,\ a\times d,\ b\times c,\ b\times d)\ ]$

where $\times$ is extended to $+\infty$ and $-\infty$ by the rule of signs:
$c \times (+\infty) = (+\infty)$ if $c > 0$, $(-\infty)$ if $c < 0$
$c \times (-\infty) = (-\infty)$ if $c > 0$, $(+\infty)$ if $c < 0$

we also need the non-standard rule: $0 \times (+\infty) = 0 \times (-\infty) = 0$

Summary of the abstract semantics

$$\mathbb{S}^\sharp[\![\,\textbf{skip}\,]\!]X^\sharp \stackrel{\text{def}}{=} X^\sharp$$

$$\mathbb{S}^\sharp[\![\,s_1; s_2\,]\!]X^\sharp \stackrel{\text{def}}{=} \mathbb{S}^\sharp[\![\,s_2\,]\!](\mathbb{S}^\sharp[\![\,s_1\,]\!]X^\sharp)$$

$$\mathbb{S}^\sharp[\![\,V \leftarrow e\,]\!]X^\sharp \stackrel{\text{def}}{=} \begin{cases} X^\sharp[V \mapsto \mathbb{E}^\sharp[\![\,e\,]\!]X^\sharp] & \text{if } \mathbb{E}^\sharp[\![\,e\,]\!]X^\sharp \neq \bot \\ \dot{\bot} & \text{if } \mathbb{E}^\sharp[\![\,e\,]\!]X^\sharp = \bot \end{cases}$$

$$\mathbb{S}^\sharp[\![\,\textbf{if } c \textbf{ then } s_1 \textbf{ else } s_2\,]\!]X^\sharp \stackrel{\text{def}}{=} \mathbb{S}^\sharp[\![\,s_1\,]\!](\mathbb{S}^\sharp[\![\,c?\,]\!]X^\sharp) \mathbin{\dot{\cup}^\sharp} \mathbb{S}^\sharp[\![\,s_2\,]\!](\mathbb{S}^\sharp[\![\,\neg c?\,]\!]X^\sharp)$$

$$\mathbb{S}^\sharp[\![\,\textbf{while } c \textbf{ do } s\,]\!]X^\sharp \stackrel{\text{def}}{=} \mathbb{S}^\sharp[\![\,\neg c?\,]\!](\text{lim } \lambda I^\sharp.\, I^\sharp \mathbin{\dot{\triangledown}} (X^\sharp \mathbin{\dot{\cup}^\sharp} \mathbb{S}^\sharp[\![\,s\,]\!](\mathbb{S}^\sharp[\![\,c?\,]\!]I^\sharp)))$$

(next slides: extending the language with assertions and local variables)

Convergence acceleration

Widening: binary operator $\nabla : \mathcal{E}^\sharp \times \mathcal{E}^\sharp \to \mathcal{E}^\sharp$ such that:

- $\gamma(X^\sharp) \cup \gamma(Y^\sharp) \subseteq \gamma(X^\sharp \nabla Y^\sharp)$ (sound abstraction of $\cup$)
- for any sequence $(X_n^\sharp)_{n\in\mathbb{N}}$, the sequence $(Y_n^\sharp)_{n\in\mathbb{N}}$

$$\begin{cases} Y_0^\sharp & \stackrel{\text{def}}{=} X_0^\sharp \\ Y_{n+1}^\sharp & \stackrel{\text{def}}{=} Y_n^\sharp \nabla X_{n+1}^\sharp \end{cases}$$

stabilizes in finite time: $\exists N \in \mathbb{N}: Y_N^\sharp = Y_{N+1}^\sharp$

Fixpoint approximation theorem:

- the sequence $X_{n+1}^\sharp \stackrel{\text{def}}{=} X_n^\sharp \nabla F^\sharp(X_n^\sharp)$ stabilizes in finite time
- when $X_{N+1}^\sharp \sqsubseteq X_N^\sharp$, then $X_N^\sharp$ abstracts lfp F

Soundness proof: assume $X_{N+1}^\sharp \sqsubseteq X_N^\sharp$, then

$\gamma(X_N^\sharp) \supseteq \gamma(X_{N+1}^\sharp) = \gamma(X_N^\sharp \nabla F^\sharp(X_N^\sharp)) \supseteq \gamma(F^\sharp(X_N^\sharp)) \supseteq F(\gamma(X_N^\sharp))$

$\gamma(X_N^\sharp)$ is a post-fixpoint of F, but lfp F is F's least post-fixpoint, so, $\gamma(X_N^\sharp) \supseteq$ lfp F

Interval widening

Interval widening $\nabla : \mathcal{I} \times \mathcal{I} \to \mathcal{I}$

$$\forall I \in \mathcal{I}: \bot \nabla I = I \nabla \bot = I$$

$$[a, b] \nabla [c, d] \stackrel{\text{def}}{=} \left[\begin{cases} a & \text{if } a \leq c \\ -\infty & \text{if } a > c \end{cases}, \begin{cases} b & \text{if } b \geq d \\ +\infty & \text{if } b < d \end{cases} \right]$$

- an unstable lower bound is put to $-\infty$
- an unstable upper bound is put to $+\infty$
- once at $-\infty$ or $+\infty$, the bound becomes stable

Point-wise lifting: $\dot{\nabla} : \mathcal{E}^\sharp \times \mathcal{E}^\sharp \to \mathcal{E}^\sharp$

$$X^\sharp \dot{\nabla} Y^\sharp \stackrel{\text{def}}{=} \lambda V \in \mathcal{V}. X^\sharp(V) \nabla Y^\sharp(V)$$

extrapolate each variable independently

$\Longrightarrow$ stabilization in at most $2|\mathcal{V}|$ iterations

Thank you for your attention.

Questions?