Formal Verification of Avionics Software

JN APMEP

David Delmas
Airbus Opérations SAS

based on slides by © A. Miné, X. Rival, and P. Cousot
École normale supérieure

20 octobre 2014

Outline

1. Introduction
2. Operational semantics
3. Denotational semantics
4. Axiomatic semantics
5. Abstract interpretation
6. Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Avionics Software for cockpit avionics computers

functions

1. **Aircraft Control Domain**: flight and embedded control systems (**C**, asm)
2. **Airline Information Services Domain**: administrative functions, flight support, and maintenance support (Java)

platforms

1. LynxOS®-based *POSIX* Host Platform for ground-board datalink applications
2. *ARINC 653* Integrated Modular Avionics
3. PikeOS®-based Avionics Server Function

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Motivation for formal verification

- Safety-critical avionics software, certified wrt DO-178 standard
  $\Rightarrow$ need for strong guarantees
- Exponential increase of size & complexity
  $\Rightarrow$ verification costs do not scale
- From process- to product-based assurance

$\Rightarrow$ Transfer of formal verification tools from academic research to operational development teams
15-year collaboration with ENS, CEA, INRIA, AbsInt

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Principle of formal verification

Define the semantics of your program

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

can be constructed from semantics of commands
of the programming language

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Principle of formal verification

Define the semantics of your program

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

can be constructed from semantics of commands
of the programming language

Define a specification

specification ≡ subset of possible behaviours

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Principle of formal verification

Define the semantics of your program

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

can be constructed from semantics of commands
of the programming language

Define a specification

specification $\equiv$ subset of possible behaviours

Conduct a formal proof

that the semantics meets the specification

use computers to automate the proof

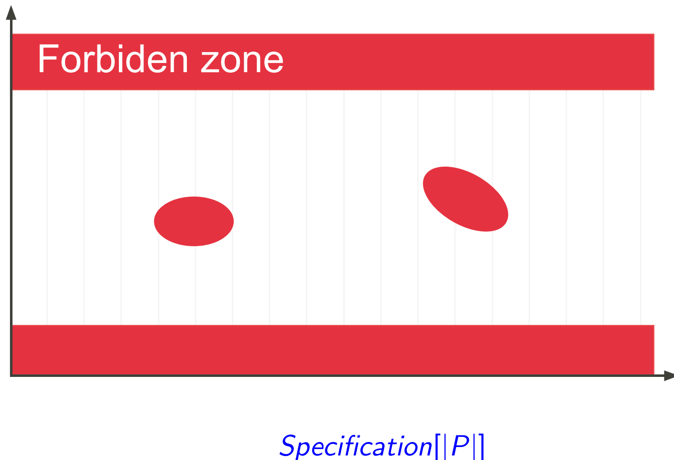
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

semantics of program P

$Semantics[|P|]$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Specification of P (e.g. safety property)



$Specification[|P|]$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Formal proof of P

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Excluded miracle

Undecidability

The semantics of a program is not computable.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Excluded miracle

Undecidability

The semantics of a program is not computable.

$\Rightarrow$ Most questions on 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
```

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Test/simulation

Subset of the possible behaviours ⇒ incomplete

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics
- Transition systems and small step semantics
- Traces semantics
  - Definitions
  - Finite traces semantics
  - Fixpoint definition
- Summary

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics
- Transition systems and small step semantics
- Traces semantics
  - Definitions
  - Finite traces semantics
  - Fixpoint definition
- Summary

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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**

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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** $\mathbb{V}$: $\mathbb{V}_{\text{int}} \cup \mathbb{V}_{\text{float}} \cup \ldots$

Syntax

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

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:

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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:**

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

A simple imperative language: semantics of conditions

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

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

$$
\begin{array}{rcl}
[\![\mathtt{TRUE}]\!](m) & = & \mathtt{TRUE} \\
[\![\mathtt{FALSE}]\!](m) & = & \mathtt{FALSE} \\
[\![\mathtt{e}_0 < \mathtt{e}_1]\!](m) & = & \begin{cases} \mathtt{TRUE} & \text{if } [\![\mathtt{e}_0]\!](m) < [\![\mathtt{e}_1]\!](m) \\ \mathtt{FALSE} & \text{if } [\![\mathtt{e}_0]\!](m) \geq [\![\mathtt{e}_1]\!](m) \\ \Omega & \text{if } [\![\mathtt{e}_0]\!](m) = \Omega \text{ or } [\![\mathtt{e}_1]\!](m) = \Omega \end{cases} \\
[\![\mathtt{e}_0 = \mathtt{e}_1]\!](m) & = & \begin{cases} \mathtt{TRUE} & \text{if } [\![\mathtt{e}_0]\!](m) = [\![\mathtt{e}_1]\!](m) \\ \mathtt{FALSE} & \text{if } [\![\mathtt{e}_0]\!](m) \neq [\![\mathtt{e}_1]\!](m) \\ \Omega & \text{if } [\![\mathtt{e}_0]\!](m) = \Omega \text{ or } [\![\mathtt{e}_1]\!](m) = \Omega \end{cases}
\end{array}
$$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

A simple imperative language: transitions

We now consider the transition induced by each statement.

Case of **assignment** $l_0 : x = \mathrm{e};\ l_1$

- if $[\![\mathrm{e}]\!](m) \neq \Omega$, then $(l_0, m) \to (l_1, m[x \leftarrow [\![\mathrm{e}]\!](m)])$
- if $[\![\mathrm{e}]\!](m) = \Omega$, then $(l_0, m) \to \Omega$

Case of **condition** $l_0 : \textbf{if}(\mathrm{c})\{l_1 : \mathrm{b}_t\ l_2\}\ \textbf{else}\{l_3 : \mathrm{b}_f\ l_4\}\ l_5$

- if $[\![\mathrm{c}]\!](m) = \mathtt{TRUE}$, then $(l_0, m) \to (l_1, m)$
- if $[\![\mathrm{c}]\!](m) = \mathtt{FALSE}$, then $(l_0, m) \to (l_3, m)$
- if $[\![\mathrm{c}]\!](m) = \Omega$, then $(l_0, m) \to \Omega$
- $(l_2, m) \to (l_5, m)$
- $(l_4, m) \to (l_5, m)$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

A simple imperative language: transitions

Case of **loop** l_0 : **while**(c){l_1 : $\mathtt{b}_t$ l_2} l_3

- if $[\![\mathtt{c}]\!](m) = \mathtt{TRUE}$, then $\begin{cases} (l_0, m) \rightarrow (l_1, m) \\ (l_2, m) \rightarrow (l_1, m) \end{cases}$
- if $[\![\mathtt{c}]\!](m) = \mathtt{FALSE}$, then $\begin{cases} (l_0, m) \rightarrow (l_3, m) \\ (l_2, m) \rightarrow (l_3, m) \end{cases}$
- if $[\![\mathtt{c}]\!](m) = \Omega$, then $\begin{cases} (l_0, m) \rightarrow \Omega \\ (l_2, m) \rightarrow \Omega \end{cases}$

Case of $\{l_0 : \mathtt{i}_0; l_1 : \ldots; l_{n-1}\mathtt{i}_{n-1}; l_n\}$

- the transition relation is defined by the individual instructions

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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:

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

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

- one instruction induces non determinism

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics
- Transition systems and small step semantics
- Traces semantics
  - Definitions
  - Finite traces semantics
  - Fixpoint definition
- Summary

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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:

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

Example:

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Example: imperative program

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

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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:

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

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:

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics
- Transition systems and small step semantics
- Traces semantics
  - Definitions
  - Finite traces semantics
  - Fixpoint definition
- Summary

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

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**

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics
- Deterministic imperative programs
- Link between operational and denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Denotation worlds

- **imperative programs**

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

  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!)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics
- Deterministic imperative programs
- Link between operational and denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

A simple imperative language: IMP

IMP expressions

$expr$	$::=$	X	(*variable*)
	$\mid$	c	(*constant*)
	$\mid$	$\diamond\ expr$	(*unary operation*)
	$\mid$	$expr \diamond expr$	(*binary operation*)

- 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: $=$, $\neq$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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])

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Expression semantics

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

$$
\begin{array}{llll}
\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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics
- Deterministic imperative programs
- Link between operational and denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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 $\mathcal{P}(\Sigma \times \Sigma_\perp)$:
  $\mathbb{N}[\![s]\!] \stackrel{\text{def}}{=} \mathbb{A}[\![s]\!] \cup \{ (\sigma, \perp) \mid \exists (\sigma_0, \ldots) \in t[\![s]\!]^\omega : \sigma = \sigma_0 \}$
  (models the terminating and non-terminating behaviors;
  can prove total correctness)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Semantic diagram

(α is an isomorphism)

denotational world | **operational world**

α

$\mathbb{S}[\![s]\!]$ denotational - - - - - - - - - - - big step $\mathbb{A}[\![s]\!]$

natural $\mathbb{N}[\![s]\!]$

traces $\approx[\![s]\!]$

transition system (small step) $\tau[s]$

statement

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics
- Specifications
- Floyd–Hoare logic
- Predicate transformers
- Conclusion

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Introduction

Operational semantics

Models precisely program execution as low-level transitions between internal states

(transition systems, execution traces, big-step semantics)

Denotational semantics

Maps programs into objects in a mathematical domain

(higher level, compositional, domain oriented)

Aximoatic semantics

Prove properties about programs

- programs are annotated with logical assertions
- a rule-system defines the validity of assertions (logical proofs)
- clearly separates programs from specifications
  (specification $\simeq$ user-provided abstraction of the behavior, it is not unique)
- enables the use of logic tools (partial automation)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics
- Specifications
- Floyd–Hoare logic
- Predicate transformers
- Conclusion

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Example: function specification

example in C + ACSL

```
int mod(int A, int B) {
  int Q = 0;
  int R = A;
  while (R >= B) {
    R = R - B;
    Q = Q + 1;
  }
  return R;
}
```

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Example: function specification

example in C + ACSL

```
//@ ensures \result == A mod B;
int mod(int A, int B) {
  int Q = 0;
  int R = A;
  while (R >= B) {
    R = R - B;
    Q = Q + 1;
  }
  return R;
}
```

- express the intended behavior of the function (returned value)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Example: function specification

example in C + ACSL

```
//@ requires A>=0 && B>=0;
//@ ensures \result == A mod B;
int mod(int A, int B) {
  int Q = 0;
  int R = A;
  while (R >= B) {
    R = R - B;
    Q = Q + 1;
  }
  return R;
}
```

- express the intended behavior of the function (returned value)
- add requirements for the function to actually behave as intended
  (a requires/ensures pair is a function contract)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Example: function specification

example in C + ACSL

```
//@ requires A>=0 && B>0;
//@ ensures \result == A mod B;
int mod(int A, int B) {
  int Q = 0;
  int R = A;
  while (R >= B) {
    R = R - B;
    Q = Q + 1;
  }
  return R;
}
```

- express the intended behavior of the function (returned value)
- add requirements for the function to actually behave as intended (a requires/ensures pair is a function contract)
- strengthen the requirements to ensure termination

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Example: program annotations

example with full assertions

```
//@ requires A>=0 && B>0;
//@ ensures \result == A mod B;
int mod(int A, int B) {
  int Q = 0;
  int R = A;
  //@ assert A>=0 && B>0 && Q=0 && R==A;
  while (R >= B) {
    //@ assert A>=0 && B>0 && R>=B && A==Q*B+R;
    R = R - B;
    Q = Q + 1;
  }
  //@ assert A>=0 && B>0 && R>=0 && R<B && A==Q*B+R;
  return R;
}
```

Assertions give detail about the internal computations
why and how contracts are fulfilled

(Note: $r = a \mod b$ means $\exists q: a = qb + r \wedge 0 \leq r < b$)

AIRBUS

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Language support

Contracts (and class invariants):

- built in few languages (Eiffel)
- available as a library / external tool (C, Java, C#, etc.)

Contracts can be:

- checked dynamically
- checked statically (Frama-C, Why, ESC/Java)
- inferred statically (CodeContracts)

In this talk:

deductive methods (logic) to check (prove) statically (at compile-time) partially automatically (with user help) that contracts hold

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics
- Specifications
- Floyd–Hoare logic
- Predicate transformers
- Conclusion

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Hoare triples

Hoare triple: $\{P\}\ prog\ \{Q\}$

- *prog* is a program fragment
- P and Q are logical assertions over program variables
  (e.g. $P \stackrel{\text{def}}{=} (X \geq 0 \wedge Y \geq 0) \vee (X < 0 \wedge Y < 0)$)

A triple means:

- if P holds before *prog* is executed
- then Q holds after the execution of *prog*
- unless *prog* does not terminates or encounters an error

P is the precondition, Q is the postcondition

$\{P\}\ prog\ \{Q\}$ expresses partial correctness
(does not rule out errors and non-termination)

Hoare triples serve as judgements in a proof system
(introduced in [Hoare69])

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Language

$$
\begin{array}{rcll}
\mathit{stat} & ::= & X \leftarrow \mathit{expr} & (\mathit{assignment}) \\
& | & \textbf{skip} & (\mathit{do\ nothing}) \\
& | & \textbf{fail} & (\mathit{error}) \\
& | & \mathit{stat};\ \mathit{stat} & (\mathit{sequence}) \\
& | & \textbf{if}\ \mathit{expr}\ \textbf{then}\ \mathit{stat}\ \textbf{else}\ \mathit{stat} & (\mathit{conditional}) \\
& | & \textbf{while}\ \mathit{expr}\ \textbf{do}\ \mathit{stat} & (\mathit{loop})
\end{array}
$$

- $X \in \mathcal{V}$: integer-valued variables
- *expr*: integer arithmetic expressions

  we assume that:

  - expressions are deterministic (for now)
  - expression evaluation do not cause error

    for instance, to avoid division by zero, we can:
    either define $1/0$ to be a valid value, such as 0
    or systematically guard divisions
    (e.g.: **if** $X = 0$ **then fail else** $\cdots/X\cdots$)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Hoare rules: summary

$$\frac{}{\{P\}\ \textbf{skip}\ \{P\}} \qquad \frac{}{\{\text{true}\}\ \textbf{fail}\ \{\text{false}\}} \qquad \frac{}{\{P[e/X]\}\ X \leftarrow e\ \{P\}}$$

$$\frac{\{P\}\ s\ \{R\} \qquad \{R\}\ t\ \{Q\}}{\{P\}\ s; t\ \{Q\}} \qquad \frac{\{P \wedge e\}\ s\ \{Q\} \qquad \{P \wedge \neg e\}\ t\ \{Q\}}{\{P\}\ \textbf{if}\ e\ \textbf{then}\ s\ \textbf{else}\ t\ \{Q\}}$$

$$\frac{\{P \wedge e\}\ s\ \{P\}}{\{P\}\ \textbf{while}\ e\ \textbf{do}\ s\ \{P \wedge \neg e\}}$$

$$\frac{P \Rightarrow P' \qquad Q' \Rightarrow Q \qquad \{P'\}\ c\ \{Q'\}}{\{P\}\ c\ \{Q\}}$$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Proof tree example

$$s \stackrel{\text{def}}{=} \textbf{while } I < N \textbf{ do } (X \leftarrow 2X;\ I \leftarrow I+1)$$

$$\dfrac{A \qquad B \qquad \dfrac{C \qquad \dfrac{}{\{P_3\}\ X \leftarrow 2X\ \{P_2\}} \qquad \dfrac{}{\{P_2\}\ I \leftarrow I+1\ \{P_1\}}}{\dfrac{\{P_1 \wedge I < N\}\ X \leftarrow 2X;\ I \leftarrow I+1\ \{P_1\}}{\{P_1\}\ s\ \{P_1 \wedge I \geq N\}}}}{\{X = 1 \wedge I = 0 \wedge N \geq 0\}\ s\ \{X = 2^N \wedge N = I \wedge N \geq 0\}}$$

$$P_1 \stackrel{\text{def}}{=} X = 2^I \wedge I \leq N \wedge N \geq 0$$

$$P_2 \stackrel{\text{def}}{=} X = 2^{I+1} \wedge I+1 \leq N \wedge N \geq 0$$

$$P_3 \stackrel{\text{def}}{=} 2X = 2^{I+1} \wedge I+1 \leq N \wedge N \geq 0 \quad \equiv X = 2^I \wedge I < N \wedge N \geq 0$$

$A : (X = 1 \wedge I = 0 \wedge N \geq 0) \Rightarrow P_1$

$B : (P_1 \wedge I \geq N) \Rightarrow (X = 2^N \wedge N = I \wedge N \geq 0)$

$C : P_3 \iff (P_1 \wedge I < N)$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Soundness and completeness

Validity:

$\{P\}\ c\ \{Q\}$ is valid $\stackrel{\text{def}}{\iff}$ executions starting in a state satisfying P
and terminating
end in a state satisfying Q

(it is an operational notion)

- **soundness**

  a proof tree exists for $\{P\}\ c\ \{Q\} \implies \{P\}\ c\ \{Q\}$ is valid

- **completeness**

  $\{P\}\ c\ \{Q\}$ is valid $\implies$ a proof tree exists for $\{P\}\ c\ \{Q\}$

  (technically, by Gödel's incompleteness theorem, $P \Rightarrow Q$ is not always provable for strong theories; hence, Hoare logic is incomplete; we consider relative completeness by adding all valid properties $P \Rightarrow Q$ on assertions as axioms)

Theorem (Cook 1974)

Hoare logic is sound (and relatively complete)

Completeness no longer holds for more complex languages (Clarke 1976)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Link with denotational semantics

Reminder: $\mathbb{S}[\![\, stat \,]\!] : \mathcal{P}(\mathcal{E}) \to \mathcal{P}(\mathcal{E})$ where $\mathcal{E} \stackrel{\text{def}}{=} \mathcal{V} \mapsto \mathcal{I}$

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

$$\mathbb{S}[\![\, \textbf{fail} \,]\!] R \stackrel{\text{def}}{=} \emptyset$$

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

$$\mathbb{S}[\![\, X \leftarrow e \,]\!] R \stackrel{\text{def}}{=} \{\, \rho[X \mapsto v] \mid \rho \in R,\ v \in \mathbb{E}[\![\, e \,]\!] \rho \,\}$$

$$\mathbb{S}[\![\, \textbf{if } e \textbf{ then } s_1 \textbf{ else } s_2 \,]\!] R \stackrel{\text{def}}{=} \mathbb{S}[\![\, s_1 \,]\!] \{\, \rho \in R \mid \text{true} \in \mathbb{E}[\![\, e \,]\!] \rho \,\} \cup \mathbb{S}[\![\, s_2 \,]\!] \{\, \rho \in R \mid \text{false} \in \mathbb{E}[\![\, e \,]\!] \rho \,\}$$

$$\mathbb{S}[\![\, \textbf{while } e \textbf{ do } s \,]\!] R \stackrel{\text{def}}{=} \{\, \rho \in \text{lfp } F \mid \text{false} \in \mathbb{E}[\![\, e \,]\!] \rho \,\}$$
$$\text{where } F(X) \stackrel{\text{def}}{=} R \cup \mathbb{S}[\![\, s \,]\!] \{\, \rho \in X \mid \text{true} \in \mathbb{E}[\![\, e \,]\!] \rho \,\}$$

Theorem

$$\{P\}\ c\ \{Q\} \stackrel{\text{def}}{\iff} \forall R \subseteq \mathcal{E}\colon R \models P \implies \mathbb{S}[\![\, c \,]\!] R \models Q$$

($A \models P$ means $\forall \rho \in A$, the formula P is true on the variable assignment ρ)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Link with denotational semantics

- Hoare logic reasons on formulas
- denotational semantics reasons on state sets

we can assimilate assertion formulas and state sets
(logical abuse: we assimilate formulas and models)

let $[R]$ be any formula representing the set R, then:

- $\{[R]\}\ c\ \{[\mathbb{S}[\![\, c \,]\!] R]\}$ is always valid
- $\{[R]\}\ c\ \{[R']\} \Rightarrow \mathbb{S}[\![\, c \,]\!] R \subseteq R'$

$\Longrightarrow$ $[\mathbb{S}[\![\, c \,]\!] R]$ provides the best valid postcondition

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics
- Specifications
- Floyd–Hoare logic
- Predicate transformers
- Conclusion

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Dijkstra's weakest liberal preconditions

Principle:

- calculus to derive preconditions from postconditions
- order and mechanize the search for intermediate assertions

  (easier to go backwards, mainly due to assignments)

Weakest liberal precondition $\quad wlp : (prog \times Prop) \rightarrow Prop$

$wlp(c, P)$ is the weakest, i.e. most general, precondition
ensuring that $\{wlp(c, P)\}\ c\ \{P\}$ is a Hoare triple

(greatest state set that ensures that the computation ends up in P)

formally: $\quad \{P\}\ c\ \{Q\} \iff (P \Rightarrow wlp(c, Q))$

"liberal" means that we do not care about termination and errors

Examples:

$wlp(X \leftarrow X + 1,\ X = 1) =$
$wlp(\textbf{while}\ X < 0\ X \leftarrow X + 1,\ X \geq 0) =$
$wlp(\textbf{while}\ X \neq 0\ X \leftarrow X + 1,\ X \geq 0) =$

(introduced in [Dijkstra75])

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Dijkstra's weakest liberal preconditions

Principle:

- calculus to derive preconditions from postconditions
- order and mechanize the search for intermediate assertions

  (easier to go backwards, mainly due to assignments)

Weakest liberal precondition $\quad wlp : (prog \times Prop) \rightarrow Prop$

$wlp(c, P)$ is the weakest, i.e. most general, precondition ensuring that $\{wlp(c, P)\}\ c\ \{P\}$ is a Hoare triple

(greatest state set that ensures that the computation ends up in P)

formally: $\quad \{P\}\ c\ \{Q\} \iff (P \Rightarrow wlp(c, Q))$

"liberal" means that we do not care about termination and errors

Examples:

$$wlp(X \leftarrow X + 1,\ X = 1) = (X = 0)$$
$$wlp(\textbf{while } X < 0\ X \leftarrow X + 1,\ X \geq 0) = \text{true}$$
$$wlp(\textbf{while } X \neq 0\ X \leftarrow X + 1,\ X \geq 0) = \text{true}$$

(introduced in [Dijkstra75])

AIRBUS

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

A calculus for wlp

wlp is defined by induction on the syntax of programs:

$$wlp(\textbf{skip}, P) \stackrel{\text{def}}{=} P$$

$$wlp(\textbf{fail}, P) \stackrel{\text{def}}{=} \text{true}$$

$$wlp(X \leftarrow e, P) \stackrel{\text{def}}{=} P[e/X]$$

$$wlp(s; t, P) \stackrel{\text{def}}{=} wlp(s, wlp(t, P))$$

$$wlp(\textbf{if } e \textbf{ then } s \textbf{ else } t, P) \stackrel{\text{def}}{=} (e \Rightarrow wlp(s, P)) \land (\neg e \Rightarrow wlp(t, P))$$

$$wlp(\textbf{while } e \textbf{ do } s, P) \stackrel{\text{def}}{=} I \land ((e \land I) \Rightarrow wlp(s, I)) \land ((\neg e \land I) \Rightarrow P)$$

- $e \Rightarrow Q$ is equivalent to $Q \lor \neg e$
  weakest property that matches Q when e holds
  but says nothing when e does not hold
- **while** loops require providing an invariant predicate I
  intuitively, *wlp* checks that I is an inductive invariant implying P
  if so, it returns I; otherwise, it returns false

wlp is the weakest precondition only if I is well-chosen. . .

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics
- Specifications
- Floyd–Hoare logic
- Predicate transformers
- Conclusion

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Conclusion

- logic allows us to reason about program correctness
- verification can be reduced to proofs of simple logic statements

Issue: automation

- annotations are required (loop invariants, contracts)
- verification conditions must be proven

to scale up to realistic programs, we need to automate as much as possible

Some solutions:

- automatic logic solvers to discharge proof obligations
  SAT / SMT solvers
- abstract interpretation to approximate the semantics
  - fully automatic
  - able to infer invariants

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation
- Principle
- Example of abstract interpretation
- Interval analysis, more formally
- Alarms
- Abstract domains

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation
- Principle
- Example of abstract interpretation
- Interval analysis, more formally
- Alarms
- Abstract domains

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Concrete semantics of program P

$x(t)$

t

Possible trajectories

Semantics$[|P|]$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Specification of P (e.g. safety property)

$Specification[|P|]$

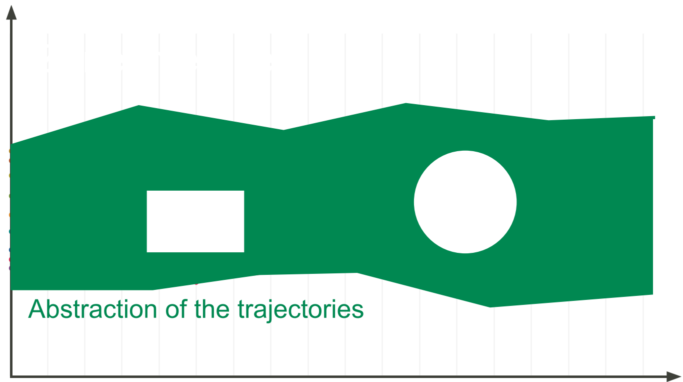
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Formal proof of P

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

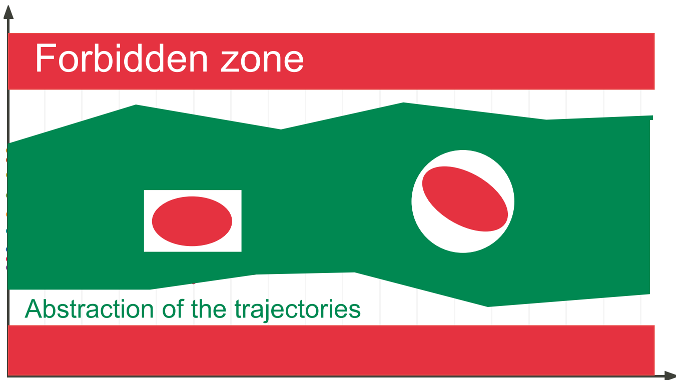
Abstract semantics for P



$Abstraction(Semantics[|P|])$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Proof by abstract interpretation



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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Soundness of abstract interpretation

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation
- Principle
- Example of abstract interpretation
- Interval analysis, more formally
- Alarms
- Abstract domains

6 Industrial state-of-the-use at Airbus

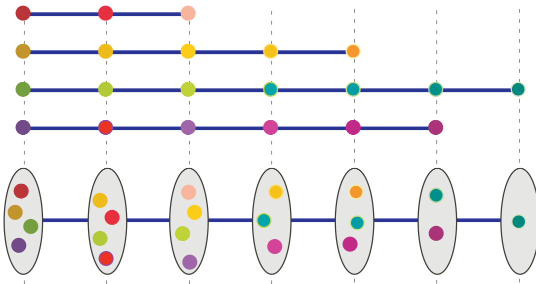
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Concrete semantics: set of (discrete) traces

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Collecting semantics

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

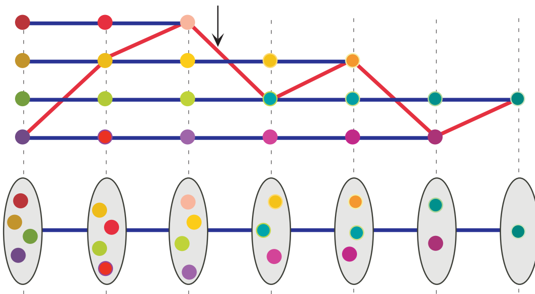


© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Trace abstraction : collecting abstraction

This an abstraction. Does the red trace exist?

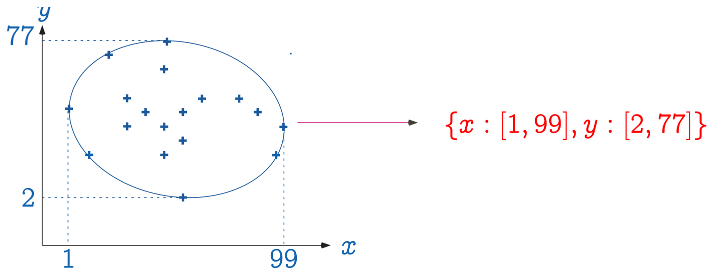
Trace semantics: no $\neq$ collecting semantics: I don't know.



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

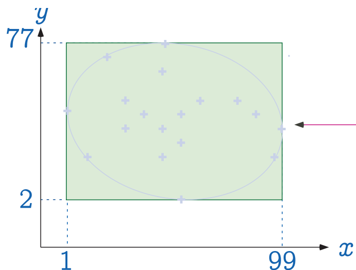
AIRBUS

Set abstraction : intervals



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Set abstraction : intervals

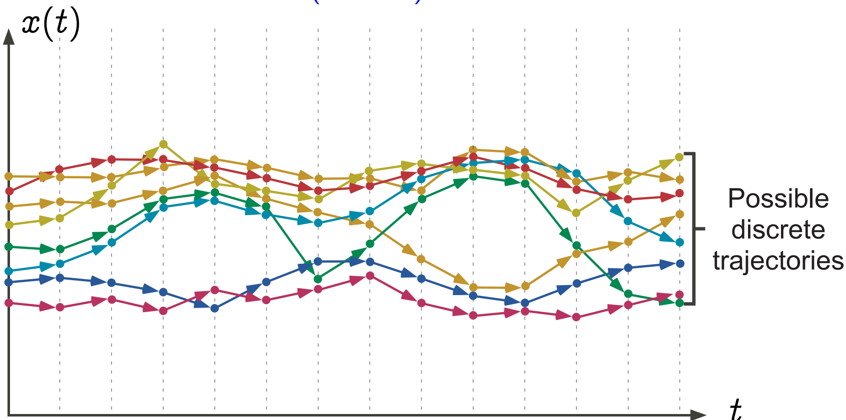


$\{x : [1, 99], y : [2, 77]\}$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
AIRBUS

From set of traces to set of reachable states

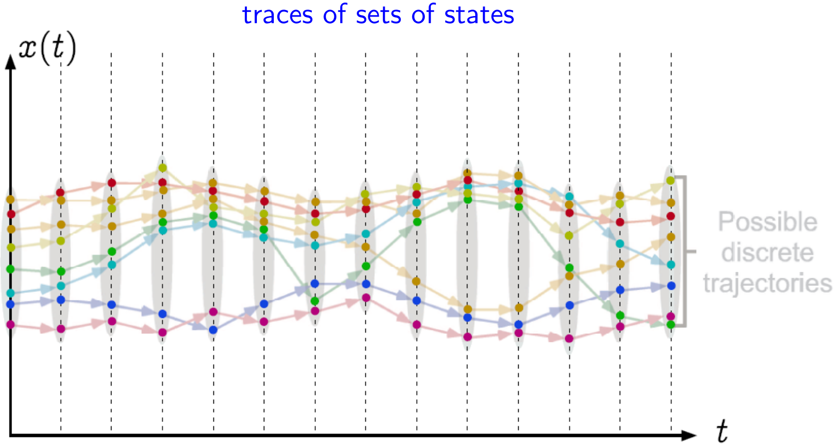
set of (discrete) traces



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

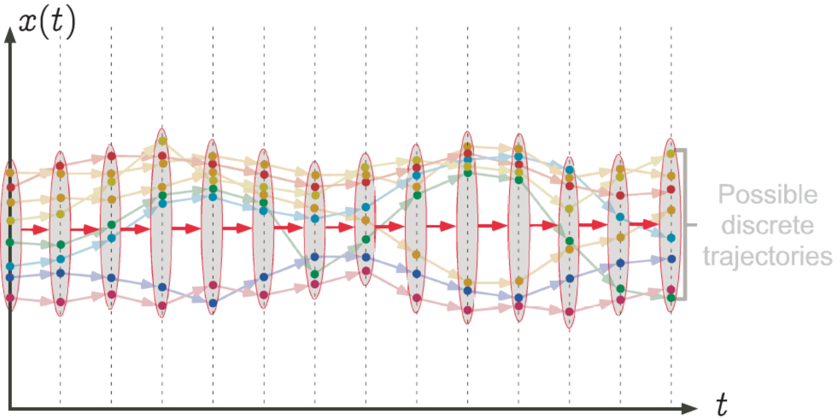
From set of traces to set of reachable states



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states

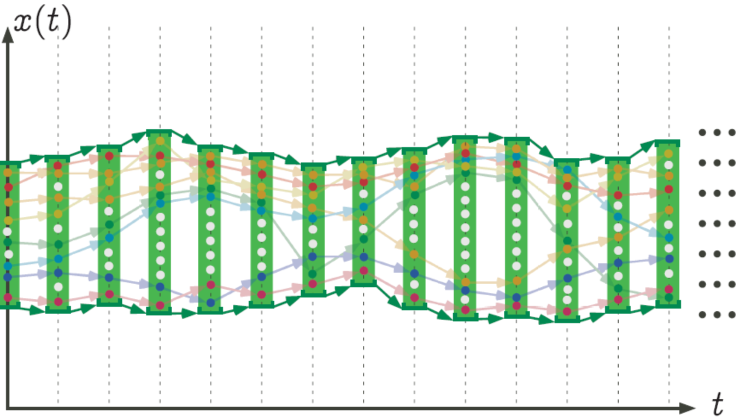
trace of sets of states



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states

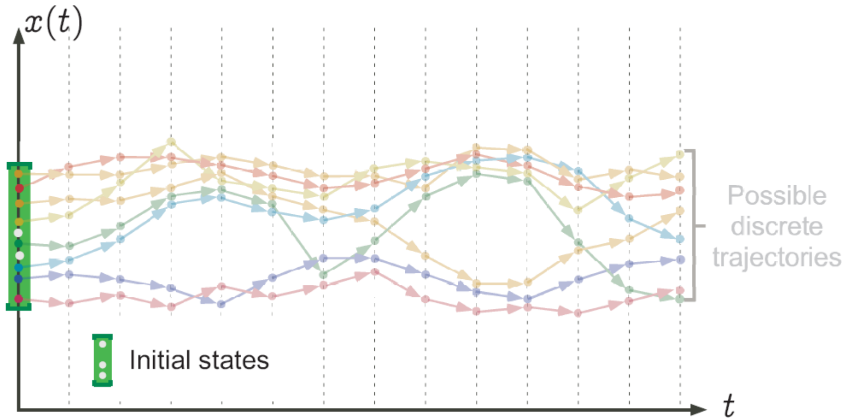
trace of intervals



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states

Effective computation : intialisation



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states

Effective computation : propagation

$x(t)$

Possible discrete trajectories

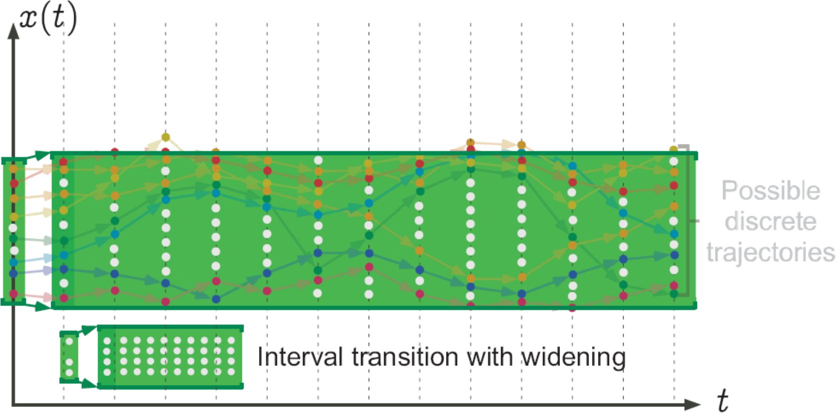
Interval transition

t

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states

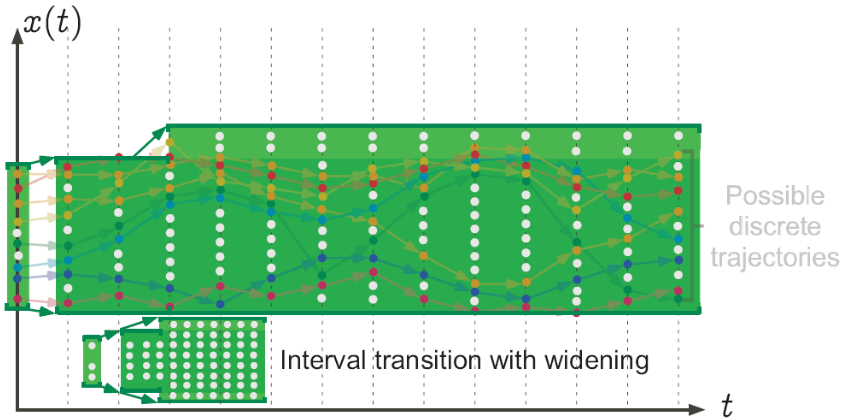
Effective computation : widening unstable constraints



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
AIRBUS

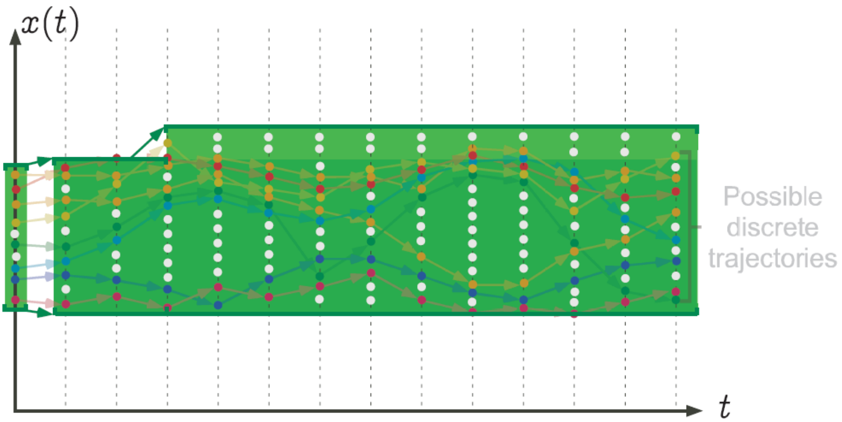
From set of traces to set of reachable states

Effective computation : widening unstable constraints



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

From set of traces to set of reachable states



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Program to be analyzed

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Equations (abstract interpretation of the semantics)

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Resolution by increasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Convergence speed-up by widening

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Interval analysis

Decreasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Decreasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Decreasing iterations

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Final solution

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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}
```

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval analysis

Formal proof of absence of overflow

```
   x := 1;
1: {x = 1}
   while x < 10000 do
2: {x ∈ [1, 9999]}          ⟵ no overflow
       x := x + 1
3: {x ∈ [2, +10000]}
   od;
4: {x = 10000}
```

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1. Introduction
2. Operational semantics
3. Denotational semantics
4. Axiomatic semantics
5. Abstract interpretation
   - Principle
   - Example of abstract interpretation
   - Interval analysis, more formally
   - Alarms
   - Abstract domains
6. Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Language

Expressions and conditions

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

Statements

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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)*

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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 $(\mathcal{P}(\mathcal{E}), \subseteq, \cup, \cap, \emptyset, \mathcal{E})$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
AIRBUS

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)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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 $\mathbb{E}[\![\, expr \,]\!]$

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval arithmetic

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

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

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

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: $0 \times (+\infty) = 0 \times (-\infty) = 0$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Summary of the abstract semantics

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

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

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

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

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

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Convergence acceleration

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

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

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

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

Fixpoint approximation theorem:

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

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

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

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

AIRBUS
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Interval widening

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

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

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

- 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: $\dot{\triangledown} : \mathcal{E}^\sharp \times \mathcal{E}^\sharp \to \mathcal{E}^\sharp$

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

extrapolate each variable independently

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

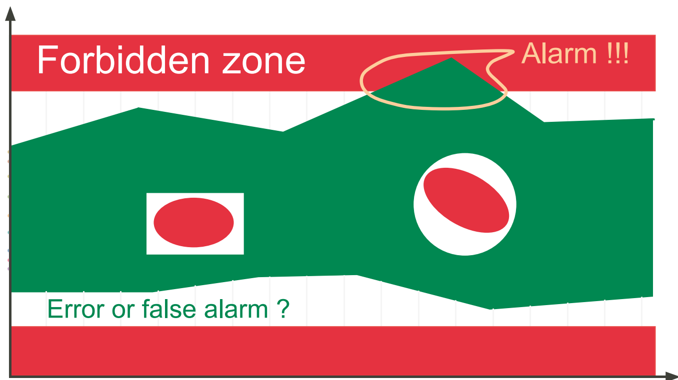
4 Axiomatic semantics

5 Abstract interpretation
- Principle
- Example of abstract interpretation
- Interval analysis, more formally
- Alarms
- Abstract domains

6 Industrial state-of-the-use at Airbus

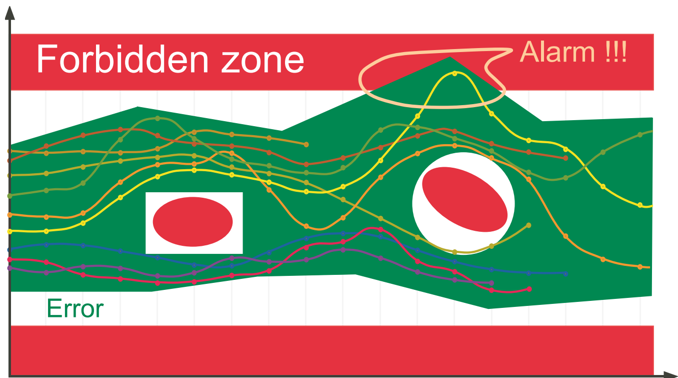
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Alarms



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

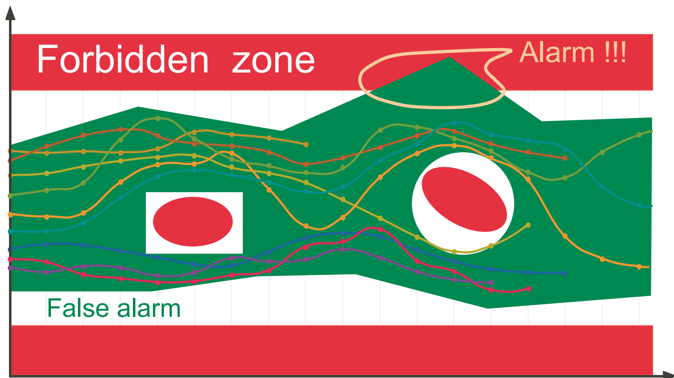
True error



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Incompleteness ⇒ false alarms



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1 Introduction

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

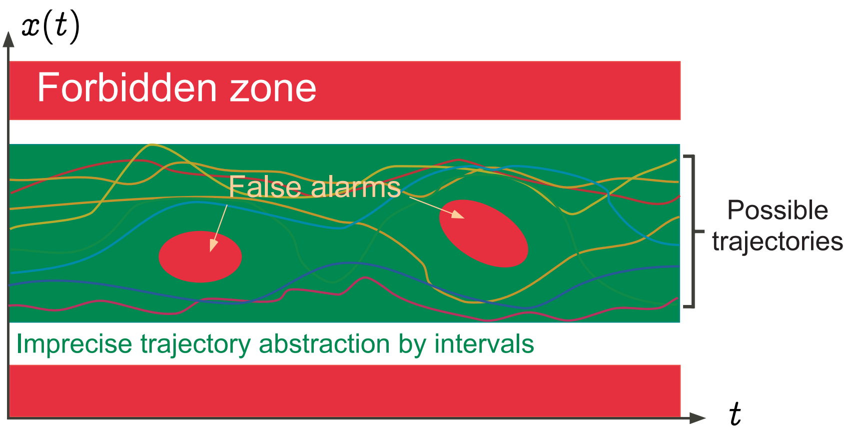
5 Abstract interpretation
- Principle
- Example of abstract interpretation
- Interval analysis, more formally
- Alarms
- Abstract domains

6 Industrial state-of-the-use at Airbus

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

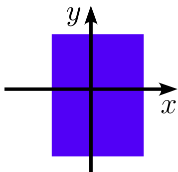
False alarms (e.g. interval analysis)



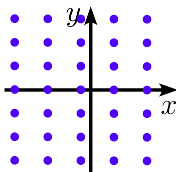
$\Rightarrow$ need for abstraction refinement

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

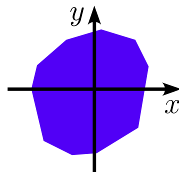
Examples of numerical abstract domains



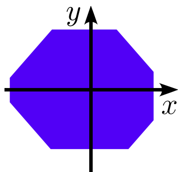
intervals
$x, y \in [a, b]$



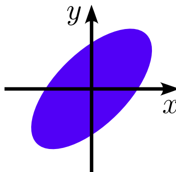
congruences
$x, y \in a\mathbb{Z} + b$



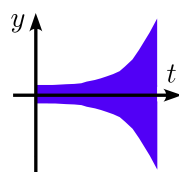
polyhedra
$\bigwedge \sum_i a_i x_i \leq b$



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



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



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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
AIRBUS

Combinations of abstractions

Semantics

(In)finite sets of (in)finite traces

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Invariant abstraction: set of reachable states

Set of points (x_i, y_i), Hoare logic

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Sign abstraction

y

x

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Interval abstraction

y

x

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Octagon abstraction

y

x

$\pm x \pm y \leq b$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Combinations of abstractions

Polyhedral abstraction

y

x

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Ellipsoidal abstraction

y

x

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Combinations of abstractions

Exponential abstraction

$$a^x \leq y$$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus
- Static analysis of source code in today's industrial processes
- Focus on run-time error analysis
  - Today: Astrée
  - Soon: AstréeA extension
- Ongoing technology transfers
- Static analysis of executables in today's industrial processes

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

- Static analysis of source code in today's industrial processes
- Focus on run-time error analysis
  - Today: Astrée
  - Soon: AstréeA extension
- Ongoing technology transfers
- Static analysis of executables in today's industrial processes

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Proof of absence of run-time errors for fly-by-wire

- AI-based static analysis of **C** source code
- ASTRÉE by AbsInt (CNRS/ENS license)
- DAL A A340/A380/A400M control software up to 650 Kloc (A350 soon)

Unit Proof on DAL A software subsets

- WP-based program proof at **C** function level
- deployed on A380/A400M/A350 fly-by-wire subsets
- Caveat (CEA) + Alt-Ergo SMT-solver (INRIA) qualified wrt. DO-178B

Data & control flow analyses

- AI-based static analysis of **C** code
- local analyses (small subsets of the call graph)
- Fan-C (Airbus), a Frama-C (CEA-INRIA) plugin

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus
- Static analysis of source code in today's industrial processes
- Focus on run-time error analysis
  - Today: Astrée
  - Soon: AstréeA extension
- Ongoing technology transfers
- Static analysis of executables in today's industrial processes

AIRBUS

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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.)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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.)

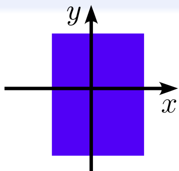
Run-time errors

- 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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

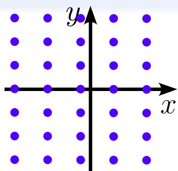
Some of ASTRÉE's numerical abstract domains

© A. Miné



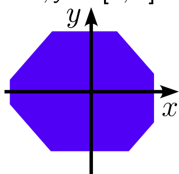
intervals

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



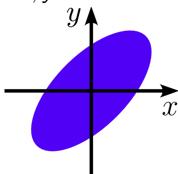
congruences

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



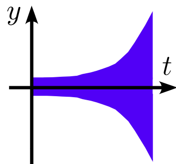
octagons

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



ellipsoids

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



geometric deviations

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

relational domains are necessary to infer precise bounds

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

The ASTRÉE static analyser

Analyseur Statique de logiciels Temps-RÉel Embarqués

A static analyzer for C programs

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

The ASTRÉE static analyser

Analyseur Statique de logiciels Temps-RÉel Embarqués

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 $\Rightarrow$ few false alarms
3. parametric $\Rightarrow$ fine-tuning by end-users

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

The ASTRÉE static analyser

Analyseur Statique de logiciels Temps-RÉel Embarqués

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

Results at Airbus

automatic proof of absence of *run-time error*
on FBW control programs of LR, A380 and A400M

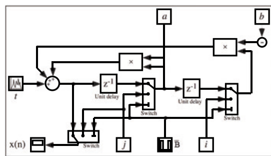
≃ 8 hours for 650 000 lines of **C**

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

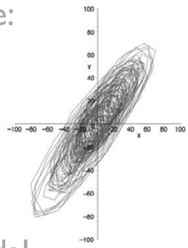
Domain-specific abstraction : ellipsoids

An abstraction for invariants of digital filters

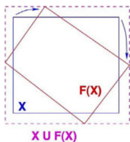
2nd order filter:



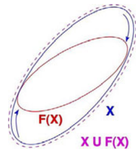
Execution trace:



Unstable polyhedral abstraction:



Stable ellipsoidal abstraction:



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

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
- but NOT deadlocks, livelocks, priority inversions

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

The ASTRÉEA prototype static analyser

An extension of ASTRÉE

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

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

The ASTRÉEA prototype static analyser

An extension of ASTRÉE

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

On-going work with Airbus

static analysis of A380 FWS

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

$\simeq$ 1000 alarms

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus

- Static analysis of source code in today's industrial processes
- Focus on run-time error analysis
  - Today: Astrée
  - Soon: AstréeA extension
- Ongoing technology transfers
- Static analysis of executables in today's industrial processes

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Numerical accuracy assessment for fly-by-wire and applications

- AI-based static analysis of **C** source code
- FLUCTUAT (CEA): successful experiments on subsets of control programs
- first target: basic numerical operators
- automate (manual) accuracy analyses

Certified compilation

- CompCert (INRIA) certified compiler
  formally verified semantic equivalence
- complementary to formal verification of **C** code
- optimizations improve WCET...
  ...while maintaining sufficient tracability
- may help save on COM/MON dissymmetry?
- requires well-defined semantics at **C** level
  $\Rightarrow$ *run-time error* analysis

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

1

2 Operational semantics

3 Denotational semantics

4 Axiomatic semantics

5 Abstract interpretation

6 Industrial state-of-the-use at Airbus
- Static analysis of source code in today's industrial processes
- Focus on run-time error analysis
  - Today: Astrée
  - Soon: AstréeA extension
- Ongoing technology transfers
- Static analysis of executables in today's industrial processes

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.
AIRBUS

AbsInt binary static analyzers used for certification

WCET Analysis on DAL A time-critical software products

- AI-based static analysis of executable code
- deployed on A340/A380/A400M/A350
  e.g. PowerPC MPC755 and 7448, TI TMS320C33
  sequential/synchronous programs, up to 650 klo**C**
- aiT WCET qualified wrt. DO-178B

Stack Analysis on all embedded software products

- AI-based static analysis of executable code
- deployed on A330/A340/A380/A400M/A350
  e.g. x86, PowerPC 755, 7448, 8610, TI TMS320C3x
  also multithreaded asynchronous programs with complex data structures up to 2 Mlo**C**
- StackAnalyzer qualified wrt. DO-178B

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Conclusion

Domaines des mathématiques dont nous avons besoin:

- Théorie des ensembles
- Ordres, théorie des treillis, points fixes
- Logique formelle, décidabilité, calculabilité
- Théorie de la preuve, théorie des types

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

7 Orderings, lattices, fixpoints
- Basic definitions on orderings
- Operators over a poset and fixpoints

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Order relations

Very useful in semantics and verification:

- **logical ordering**, expresses **implication** of logical facts
- **computational ordering**, useful to establish well-foundedness of fixpoint definitions, and for termination

Definition: partially ordered set (poset)

Let a set $\mathcal{S}$ and a binary relation $\sqsubseteq \subseteq \mathcal{S} \times \mathcal{S}$ over $\mathcal{S}$. Then, $\sqsubseteq$ is an **order relation** (and $(\mathcal{S}, \sqsubseteq)$ is called a **poset**) if and only if it is

- **reflexive:** $\forall x \in \mathcal{S},\ x \sqsubseteq x$
- **transitive:** $\forall x, y, z \in \mathcal{S},\ x \sqsubseteq y \wedge y \sqsubseteq z \implies x \sqsubseteq z$
- **antisymmetric:** $\forall x, y \in \mathcal{S},\ x \sqsubseteq y \wedge y \sqsubseteq x \implies x = y$

- notation: $x \sqsubset y ::= (x \sqsubseteq y \wedge x \neq y)$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Graphical representation

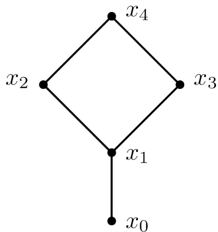
We often use **Hasse diagrams** to represent posets:

Extensive definition:

- $\mathcal{S} = \{x_0, x_1, x_2, x_3, x_4\}$
- $\sqsubset$ defined by:

$$
\begin{aligned}
x_0 &\sqsubset x_1 \\
x_1 &\sqsubset x_2 \\
x_1 &\sqsubset x_3 \\
x_2 &\sqsubset x_4 \\
x_3 &\sqsubset x_4
\end{aligned}
$$

Diagram:



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Total ordering

Definition: total order relation

Order relation $\sqsubseteq$ over $\mathcal{S}$ is a **total** order if and only if

$$\forall x, y \in \mathcal{S},\ x \sqsubseteq y \vee y \sqsubseteq x$$

- $(\mathbb{R}, \leq)$ is a total ordering
- if set $\mathcal{S}$ has at least two distinct elements x, y then its powerset $(\mathcal{P}(\mathcal{S}), \subseteq)$ is **not** a total order
  indeed $\{x\}, \{y\}$ cannot be compared
- most of the order relations we will use are *not* be total

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Minimum and maximum elements

Definition: extremal elements

Let $(\mathcal{S}, \sqsubseteq)$ be a poset and $\mathcal{S}' \subseteq \mathcal{S}$. Then x is

- **minimum element** of $\mathcal{S}'$ if and only if
  $x \in \mathcal{S}' \land \forall y \in \mathcal{S}',\ x \sqsubseteq y$
- **maximum element** of $\mathcal{S}'$ if and only if
  $x \in \mathcal{S}' \land \forall y \in \mathcal{S}',\ y \sqsubseteq x$

- maximum and minimum elements **may not exist**
  example: $\{\{x\}, \{y\}\}$ in the powerset, where $x \neq y$
- **infimum $\perp$ ("bottom"):** minimum element of $\mathcal{S}$
- **supremum $\top$ ("top"):** maximum element of $\mathcal{S}$

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Upper bounds and least upper bound

Definition: bounds

Given poset $(\mathcal{S}, \sqsubseteq)$ and $\mathcal{S}' \subseteq \mathcal{S}$, then $x \in \mathcal{S}$ is

- an **upper bound** of $\mathcal{S}'$ if

$$\forall y \in \mathcal{S}',\ y \sqsubseteq x$$

- the **least upper bound** (lub) of $\mathcal{S}'$ (noted $\sqcup \mathcal{S}'$) if

$$\forall y \in \mathcal{S}',\ y \sqsubseteq x \wedge \forall z \in \mathcal{S}, (\forall y \in \mathcal{S}',\ y \sqsubseteq z) \implies x \sqsubseteq z$$

- if it exists, the least upper bound is **unique**:
  if x, y are least upper bounds of $\mathcal{S}$, then $x \sqsubseteq y$ and $y \sqsubseteq x$,
  thus $x = y$ by antisymmetry
- notation: $x \sqcup y ::= \sqcup\{x, y\}$
- upper bounds and least upper bounds **may not exist**
- **dual notions:** lower bound, greatest lower bound (glb, noted $\sqcap \mathcal{S}'$)

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Duality principle

So far all definitions admit a symmetric counterpart

- given an order relation $\sqsubseteq$, $\mathcal{R}$ defined by $x \mathcal{R} y \iff y \sqsubseteq x$ is also an order relation
- thus all properties that can be proved about $\sqsubseteq$ also have a symmetric property that also holds

This is the **duality principle**:

minimum element	maximum element
infimum	supremum
lower bound	upper bound
greatest lower bound	least upper bound

... more to follow

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Complete lattice

Definition: complete lattice

A **complete lattice** is a tuple $(\mathcal{S}, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ where:

- $(\mathcal{S}, \sqsubseteq)$ is a poset
- $\bot$ is the infimum of $\mathcal{S}$
- $\top$ is the supremum of $\mathcal{S}$
- any subset $\mathcal{S}'$ of $\mathcal{S}$ has a lub $\sqcup \mathcal{S}'$ and a glb $\sqcap \mathcal{S}'$

Properties:

- $\bot = \sqcup \emptyset = \sqcap \mathcal{S}$
- $\top = \sqcap \emptyset = \sqcup \mathcal{S}$

Example: the **powerset** $(\mathcal{P}(\mathcal{S}), \subseteq, \emptyset, \mathcal{S}, \cup, \cap)$ of set $\mathcal{S}$ is a complete lattice

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Lattice

The existence of lubs and glbs for all subsets is often a very strong property, that may not be met:

Definition: lattice

A **lattice** is a tuple $(\mathcal{S}, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ where:

- $(\mathcal{S}, \sqsubseteq)$ is a poset
- $\bot$ is the infimum of $\mathcal{S}$
- $\top$ is the supremum of $\mathcal{S}$
- any **pair** $\{x, y\}$ of $\mathcal{S}$ has a lub $x \sqcup y$ and a glb $x \sqcap y$

- let $\mathcal{Q} = \{q \in \mathbb{Q} \mid 0 \leq q \leq 1\}$;
  then $(\mathcal{Q}, \leq)$ is a **lattice** but **not a complete lattice**
  indeed, $\{q \in \mathcal{Q} \mid q \leq \frac{\sqrt{2}}{2}\}$ has no lub in $\mathcal{Q}$
- property: a **finite** lattice is also a complete lattice

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Chains

Definition: increasing chain

Let $(\mathcal{S}, \sqsubseteq)$ be a poset and $\mathcal{C} \subseteq \mathcal{S}$.
It is an **increasing chain** if and only if

- it has an infimum
- poset $(\mathcal{C}, \sqsubseteq)$ is total (i.e., any two elements can be compared)

Example, in the powerset $(\mathcal{P}(\mathbb{N}), \subseteq)$:

$$\mathcal{C} = \{c_i \mid i \in \mathbb{N}\} \quad \text{where} \quad c_i = \{2^0, 2^2, \ldots, 2^i\}$$

Definition: increasing chain condition

Poset $(\mathcal{S}, \sqsubseteq)$ **satisfies the increasing chain condition** if and only if any increasing chain $\mathcal{C} \subseteq \mathcal{S}$ is finite.

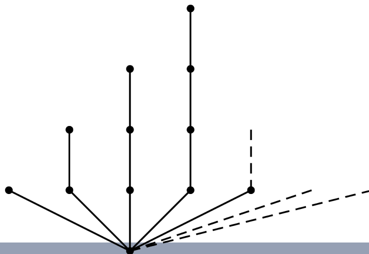
© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Complete partial orders

Definition: complete partial order

A **complete partial order** (cpo) is a poset $(\mathcal{S}, \sqsubseteq)$ such that any increasing chain $\mathcal{C}$ of $\mathcal{S}$ has a least upper bound. A **pointed cpo** is a cpo with an infimum $\perp$.

- clearly, any complete lattice is a cpo
- the opposite is not true:



© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Outline

7 Orderings, lattices, fixpoints
- Basic definitions on orderings
- Operators over a poset and fixpoints

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Operators over a poset

Definition: operators and orderings

Let $(\mathcal{S}, \sqsubseteq)$ be a poset and $\phi : \mathcal{S} \to \mathcal{S}$ be an operator over $\mathcal{S}$.
Then, ϕ is:

- **monotone** if and only if $\forall x, y \in \mathcal{S},\ x \sqsubseteq y \Longrightarrow \phi(x) \sqsubseteq \phi(y)$
- **continuous** if and only if, for any chain $\mathcal{S}' \subseteq \mathcal{S}$ then:
  $$\begin{cases} \text{if } \sqcup \mathcal{S}' \text{ exists, so does } \sqcup \{\phi(x) \mid x \in \mathcal{S}'\} \\ \text{and } \phi(\sqcup \mathcal{S}') = \sqcup\{\phi(x) \mid x \in \mathcal{S}'\} \end{cases}$$
- **$\sqcup$-preserving** if and only if:
  $$\forall \mathcal{S}' \subseteq \mathcal{S}, \begin{cases} \text{if } \sqcup \mathcal{S}' \text{ exists, then } \sqcup \{\phi(x) \mid x \in \mathcal{S}'\} \text{ exists} \\ \text{and } \phi(\sqcup \mathcal{S}') = \sqcup\{\phi(x) \mid x \in \mathcal{S}'\} \end{cases}$$

Notes:

- "monotone" in English means "*croissante*" in French ; "*décroissante*" translates into "anti-monotone" and "monotone" into "*isotone*"

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Operators over a poset

A few interesting properties:

- **continuous $\Rightarrow$ monotone:**
  if ϕ is monotone, and $x, y \in \mathcal{S}$ are such that $x \sqsubseteq y$, then $\{x, y\}$ is a chain with lub y, thus $\phi(x) \sqcup \phi(y)$ exists and is equal to $\phi(\sqcup\{x, y\}) = \phi(y)$; therefore $\phi(x) \sqsubseteq \phi(y)$.
- **$\sqcup$-preserving $\Rightarrow$ monotone:**
  same argument.

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Fixpoints

Definition: fixpoints

Let $(\mathcal{S}, \sqsubseteq)$ be a poset and $f : \mathcal{S} \to \mathcal{S}$ be an operator over $\mathcal{S}$.

- a **fixpoint** of ϕ is an element x such that $\phi(x) = x$
- a **pre-fixpoint** of ϕ is an element x such that $x \sqsubseteq \phi(x)$
- a **post-fixpoint** of ϕ is an element x such that $\phi(x) \sqsubseteq x$
- the **least fixpoint** $\operatorname{lfp} \phi$ of ϕ (if it exists, it is unique) is the smallest fixpoint of ϕ
- the **greatest fixpoint** $\operatorname{gfp} \phi$ of ϕ (if it exists, it is unique) is the greatest fixpoint of ϕ

Note: the existence of a least fixpoint, a greatest fixpoint or even a fixpoint is *not guaranteed*; we will see several theorems that establish their existence under specific assumptions...

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Tarski's Theorem

Theorem

Let $(\mathcal{S}, \sqsubseteq, \bot, \top, \sqcup, \sqcap)$ be a complete lattice and $\phi : \mathcal{S} \to \mathcal{S}$ be a monotone operator over $\mathcal{S}$. Then:

1. ϕ has a least fixpoint $\text{lfp}\,\phi$ and $\text{lfp}\,\phi = \sqcap\{x \in \mathcal{S} \mid \phi(x) \sqsubseteq x\}$.
2. ϕ has a greatest fixpoint $\text{gfp}\,\phi$ and $\text{gfp}\,\phi = \sqcup\{x \in \mathcal{S} \mid x \sqsubseteq \phi(x)\}$.
3. the set of fixpoints of ϕ is a complete lattice.

Proof of point 1:

We let $X = \{x \in \mathcal{S} \mid \phi(x) \sqsubseteq x\}$ and $x_0 = \sqcap X$.

Let $y \in X$:

- $x_0 \sqsubseteq y$ by definition of the glb;
- thus, since ϕ is monotone, $\phi(x_0) \sqsubseteq \phi(y)$;
- thus, $\phi(x_0) \sqsubseteq y$ since $\phi(y) \sqsubseteq y$, by definition of X.

Therefore $\phi(x_0) \sqsubseteq x_0$, since $x_0 = \sqcap X$.

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS

Tarski's Theorem

We proved that $\phi(x_0) \sqsubseteq x_0$. We derive from this that:

- $\phi(\phi(x_0)) \sqsubseteq \phi(x_0)$ since ϕ is monotone;
- $\phi(x_0)$ is a post-fixpoint of ϕ, thus $\phi(x_0) \in X$;
- $x_0 \sqsubseteq \phi(x_0)$ by definition of the greatest lower bound

We have established both inclusions so $\phi(x_0) = x_0$.

Proof of point 2: similar, by duality.

Proof of point 3:

- if X is a set of fixpoints of ϕ, we need to consider ϕ over $\{y \in \mathcal{S} \mid y \sqsubseteq_{\mathcal{S}} \sqcap X\}$ to establish the existence of a **glb of X in the poset of fixpoints**
- the existence of **least upper bounds in the poset of fixpoints** follows by duality

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Kleene's Theorem

Tarski's theorem guarantees existence of an lfp, but is not constructive.

Theorem

Let $(\mathcal{S}, \sqsubseteq, \bot)$ be a pointed cpo and $\phi : \mathcal{S} \rightarrow \mathcal{S}$ be a continuous operator over $\mathcal{S}$. Then ϕ has a least fixpoint, and

$$\mathrm{lfp}\,\phi = \bigsqcup_{n \in \mathbb{N}} \phi^n(\bot)$$

First, we prove **the existence of the lub**:

Since ϕ is continuous, it is also monotone. We can prove by induction over n that $\{\phi^n(\bot) \mid n \in \mathbb{N}\}$ is a chain:

- $\phi^0(\bot) = \bot \sqsubseteq \phi(\bot)$ by definition of the infimum;
- if $\phi^n(\bot) \sqsubseteq \phi^{n+1}(\bot)$, then $\phi^{n+1}(\bot) = \phi(\phi^n(\bot)) \sqsubseteq \phi(\phi^{n+1}(\bot)) = \phi^{n+2}(\bot)$

By definition of the cpo structure, the lub exists. We let x_0 denote it.

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Kleene's Theorem

Secondly, we prove that **it is a fixpoint of ϕ**:

Since ϕ is continuous, $\{\phi^{n+1}(\bot) \mid n \in \mathbb{N}\}$ has a lub, and

$$\begin{array}{rcll} \phi(x_0) & = & \phi(\sqcup\{\phi^n(\bot) \mid n \in \mathbb{N}\}) & \\ & = & \{\sqcup\phi^{n+1}(\bot) \mid n \in \mathbb{N}\} & \text{by continuity of } \phi \\ & = & \bot \sqcup \{\sqcup\phi^{n+1}(\bot) \mid n \in \mathbb{N}\} & \text{by definition of } \bot \\ & = & x_0 & \text{by simple rewrite} \end{array}$$

Last, we show that it is the **least** fixpoint:

Let x_1 denote another fixpoint of ϕ. We show by induction over n that $\phi^n(\bot) \sqsubseteq x_1$:

- $\phi^0(\bot) = \bot \sqsubseteq x_1$ by definition of $\bot$;
- if $\phi^n(\bot) \sqsubseteq x_1$, then $\phi^{n+1}(\bot) \sqsubseteq \phi(x_1) = x_1$ by monotony, and since x_1 is a fixpoint.

By definition of the lub, $x_0 \sqsubseteq x_1$

AIRBUS

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Automata example, constructive

We can now state a **constructive definition** of the automaton semantics. Operator ϕ is defined by

$$\phi(f) = \lambda(q \in Q) \cdot \begin{cases} \{\epsilon\} \cup \phi_0(f)(q_\mathrm{i}) & \text{if } q = q_\mathrm{i} \\ \phi_0(f)(q) & \text{otherwise} \end{cases}$$

Proof steps:

- ϕ is continuous
- thus, Kleene's theorem applies so $\operatorname{lfp} \phi$ exists and $\operatorname{lfp} \phi = \bigcup_{n \in \mathbb{N}} \phi^n(\bot)$...
  ... this actually saves the double inclusion proof to establish that $[\![\mathcal{A}]\!] = \operatorname{lfp} \phi$

Furthermore, $[\![\mathcal{A}]\!] = \bigcup_{n \in \mathbb{N}} \phi^n(\bot)$.

This fixpoint definition will be very useful to infer or verify semantic properties.

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

Duality principle

We can extend the duality notion:

monotone	monotone
anti-monotone	anti-monotone
post-fixpoint	pre-fixpoint
least fixpoint	greatest fixpoint
increasing chain	decreasing chain

Furthermore both Tarski's theorem and Kleene's theorem have a dual version (Tarski's theorem mostly encloses its own dual, except for the definition of the gfp).

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

or the disclosure of its content. This document shall not be reproduced or disclosed to a third party without the express written consent of AIRBUS Operations S.A.S. This document and its content shall not be used for any purpose other than that for which it is supplied. The statements made herein do not constitute an offer. They are based on the mentioned assumptions and are expressed in good faith. Where the supporting grounds for these statements are not shown, AIRBUS Operations S.A.S will be pleased to explain the basis thereof. AIRBUS, its logo, A300, A310, A318, A319, A320, A321, A330, A340, A350, A380, A400M are registered trademarks.

Thank you for your attention.

If you are interested, please contact

david.delmas@airbus.com

Questions?

© AIRBUS Operations S.A.S. All rights reserved. Confidential and proprietary document.

AIRBUS