Semantic Formalisms: an overview

Eric Madelaine
eric.madelaine@sophia.inria.fr

INRIA Sophia-Antipolis
Oasis team

Mastère Réseaux et Systèmes Distribués
TC4

Program of the course: 1: Semantic Formalisms

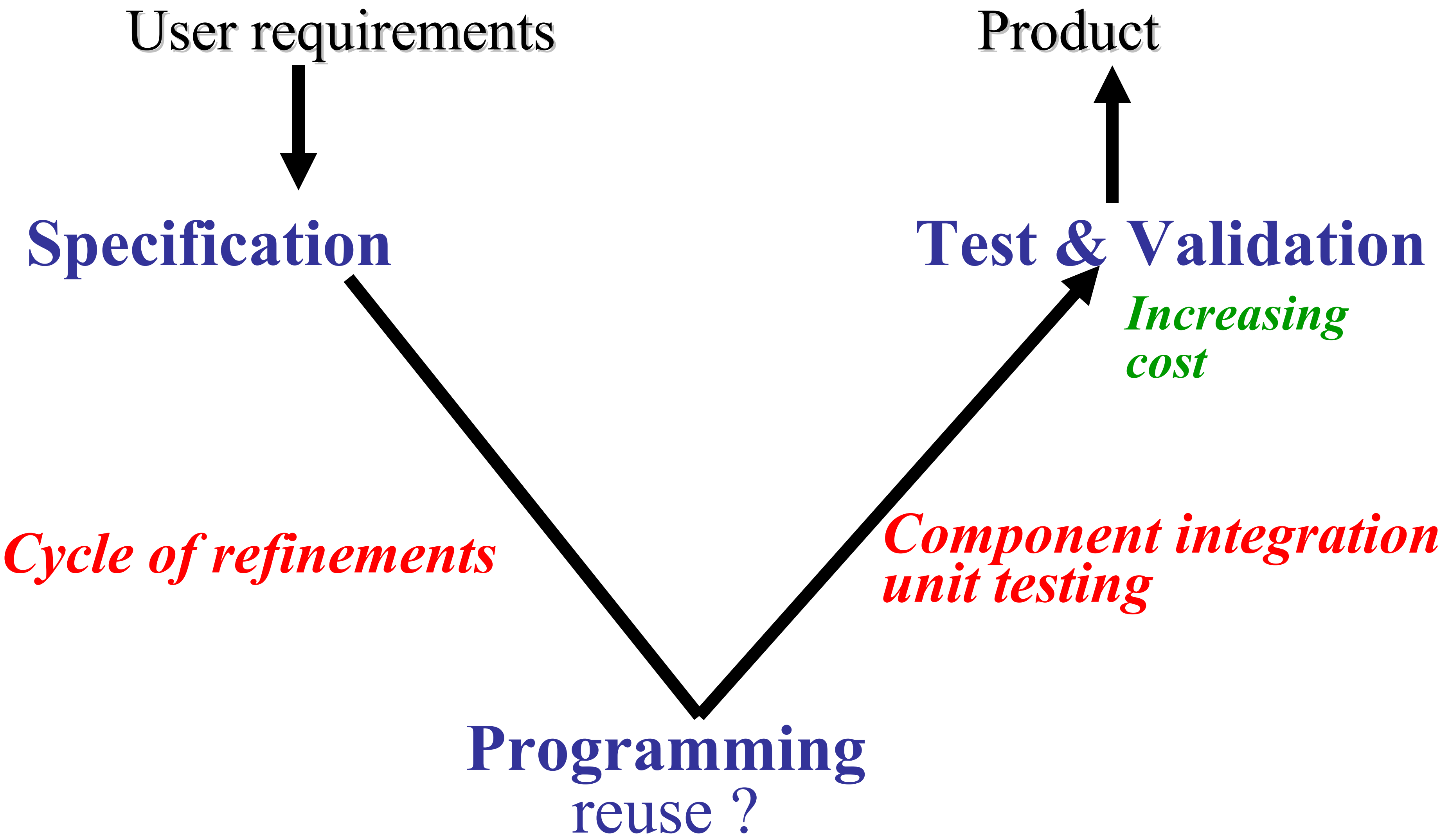
- Semantics and formal methods:
  - motivations, definitions, examples
- Denotational semantics : give a precise meaning to programs
  - abstract interpretation
- Operational semantics, behaviour models : represent the complete behaviour of the system
  - CCS, Labelled Transition Systems

Goals of (semi) Formal Methods

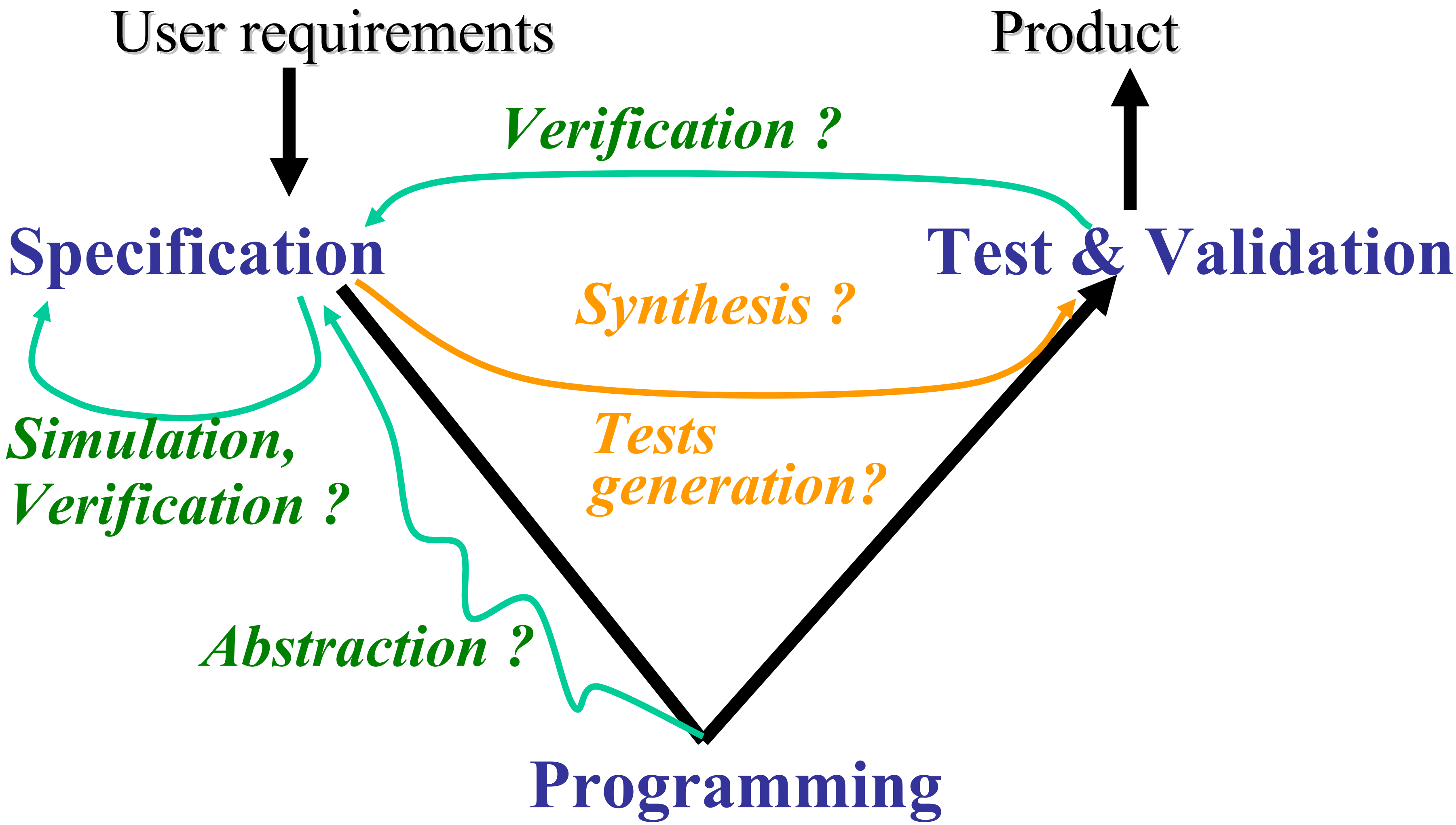
- Develop programs and systems as mathematical objects
- Represent them (syntax)
- Interpret/Execute them (semantics)
- Analyze / reason about their behaviours
  (algorithmic, complexity, verification)
- In addition to debug, using exhaustive tests and property checking.

Software engineering (ideal view)

- Requirements informal
  - User needs, general functionalities.
  - incomplete, unsound, *open*
- Detailed specification formal ?
  - Norms, standards?..., at least a reference
  - Separation of architecture and function. *No ambiguities*
- development
  - Practical implementation of components
  - Integration, deployment
- Tests (units then global) *vs* verification ?
  - Experimental simulations, certification



V cycle (utopia)



Benefits from formal methods ?
automatisation?

Support UML (aparté)

- Notation standardisée, une profusion de modèles/diagrammes :
  - class diagrams
  - use-case diagrams
  - séquence diagrams
  - statecharts et activity charts
  - deployment diagrams
- + stéréotypes pour particulariser les modèles (UML-RT, Embedded UML, …)
- Sémantique ? Flot de conception et méthodologie?

Developer Needs

- Notations, syntax
  - textual
  - graphical (charts, diagrams…)
- Meaning, semantics
  - Non ambiguous signification, executability
  - interoperability, standards
- Instrumentation analysis methods
  - prototyping, light-weight simulation
  - verification

How practical is this ?

- Currently an utopia for large software projects, but :
  - Embedded systems
    - Safety is essential (no possible correction)
  - Critical systems
    - Safety, human lives (travel, nuclear)

      Ligne Meteor, Airbus, route intelligente
    - Safety, economy (e-commerce, cost of bugs)

      Panne réseau téléphonique US, Ariane 5
    - Safety, large volume (microprocessors)

      Bug Pentium

Industry succes-stories

- Model-checking for circuit development
  - Finite systems, mixing combinatory logics with register states
- Specification of telecom standards
- Proofs of Security properties for Java code and crypto-protocols.
- Certification of embedded software (trains, aircafts)
- Synthesis ?

Semantics: definition, motivations

- **Give a (formal) meaning to words, objects, sentences, programs…**

Why ?

- Natural language specifications are not sufficient
- A need for understanding languages: eliminate ambiguities, get a better confidence.
- Precise, compact and complete definition.
- Facilitate learning and implementation of languages

Formal semantics, Proofs, and Tools

- **Manual proofs are error-prone !**
- **Tools for Execution and Reasoning**
  - semantic definitions are input for meta-tools
- **Integrated in the development cycle**
  - consistent and safe specifications
  - requires validation (proofs, tests, …)
- **Challenge:**

Expressive power versus executability...

Concrete syntax, Abstract syntax, and Semantics

- **Concrete syntax:**
  - scanners, parsers, BNF, ... many tools and standards.
- **Abstract syntax:**
  - operators, types, => *tree representations*
- **Semantics:**
  - based on abstract syntax
  - static semantics: typing, analysis, transformations
  - dynamic: evaluation, behaviours, ...

This is not only a concern for theoreticians: it is the very basis for compilers, programming environments, testing tools, etc...

Static semantics : examples

Checks non-syntactic constraints

- compiler front-end :
  - declaration and utilisation of variables,
  - typing, scoping, … static typing => no execution errors ???
- or back-ends :
  - optimisers
- defines legal programs :
  - Java byte-code *verifier*
  - JavaCard: legal acces to shared variables through firewall

Dynamic semantics

- Gives a meaning to the program (a semantic value)
- Describes the behaviour of a (legal) program
- Defines a language interpreter

  |- e -> e '

  let i=3 in 2*i -> semantic value = 6

- Describes the properties of legal programs

The different semantic families (1)

- **Denotational semantics**
  - mathematical model, high level, abstract
- **Axiomatic semantics**
  - provides the language with a theory for proving properties / assertions of programs
- **Operational semantics**
  - computation of the successive states of an abstract machine.

Semantic families (2)

- **Denotational semantics**
  - defines a model, an abstraction, an interpretation

    $\Rightarrow$ *for the language designers*
- **Axiomatic semantics**
  - builds a logical theory

    $\Rightarrow$ *for the programmers*
- **Operational semantics**
  - builds an interpreter, or a finite representation

    $\Rightarrow$ *for the language implementors*

Semantic families (3) relations between :

- denotational / operational
  – implementation correct wrt model

- axiomatic / denotational
  – completeness of the theory wrt the model

Program of the course: 1: Semantic Formalisms

- Semantics and formal methods:
  - motivations, definitions, examples
- Denotational semantics : give a precise meaning to programs
  - abstract interpretation
- Operational semantics, behaviour models : represent the complete behaviour of the system
  - CCS, Labelled Transition Systems

Denotational semantics

- Gives a **mathematical model** (interpretation)
  for any program of a language.
  All possible computations in all possible environments
  Examples of domains:
  lambda-calculus, high-level functions, pi-calculus, etc...

- Different **levels of precision** : hierarchy of semantics, related by abstraction.
- When coarse enough
  => **effectively computable (finite representation)**
  (automatic) static analysis.

Abstract Interpretation

- **Motivations** :
  - Analyse complex systems by reasoning on simpler models.
  - Design models that preserve the desired properties
  - Complete analysis is undecidable
- **Abstract domains :**
  - abstract properties (sets), abstract operations
  - Galois connections: relate domains by adequate abstraction/concretisation functions.

Abstract Interpretation (2)

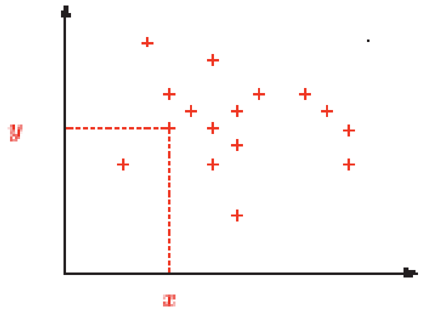
- **Example :**
  - Program with 2 integer variables X and Y
  - Trace semantics = all possible computation traces (sequences of states with values of X and Y)
  - Collecting semantics =

    (infinite) set of values of pairs <x,y>
  - Further Abstractions :

    Signs : N --> {-,0,+}

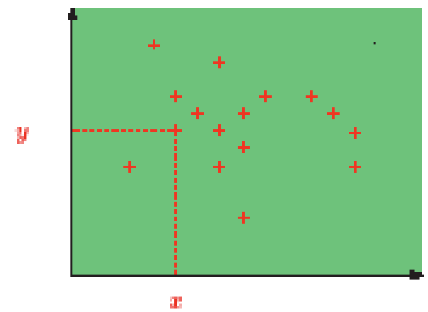
    succ --> - --> {-,0}
0 --> +
+ --> +

Abstract Interpretation (3)



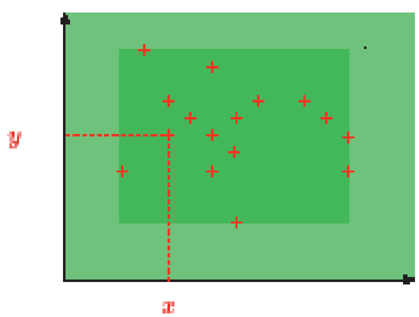
$\{\ldots, \langle 5, 7\rangle, \ldots, \langle 13, 21\rangle, \ldots\}$

(a) [In]finite Set of Points



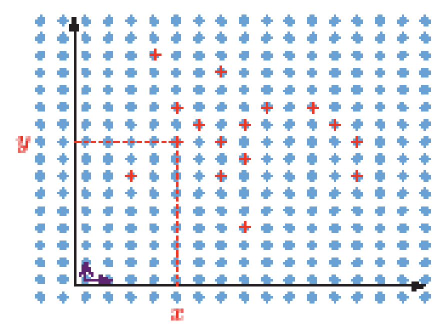
$\begin{cases} x \geq 0 \\ y \geq 0 \end{cases}$

(b) Sign Abstraction



$\begin{cases} x \in [3,\ 27] \\ y \in [4,\ 32] \end{cases}$

(c) Interval Abstraction

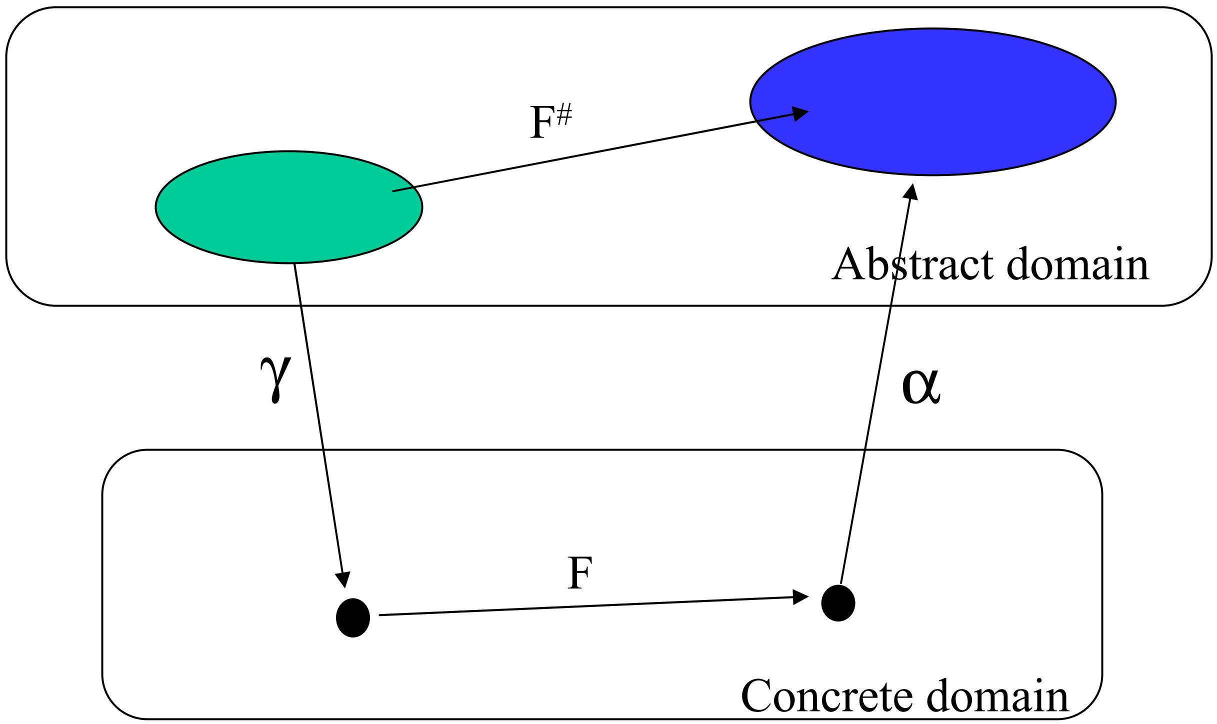


$\begin{cases} x = 5 \bmod 8 \\ y = 7 \bmod 9 \end{cases}$

(d) Simple Congruence Abstraction

Abstract Interpretation (4)

– Function Abstraction: $F^{\#} = \gamma \circ F \circ \alpha$



Abstract Interpretation (5)

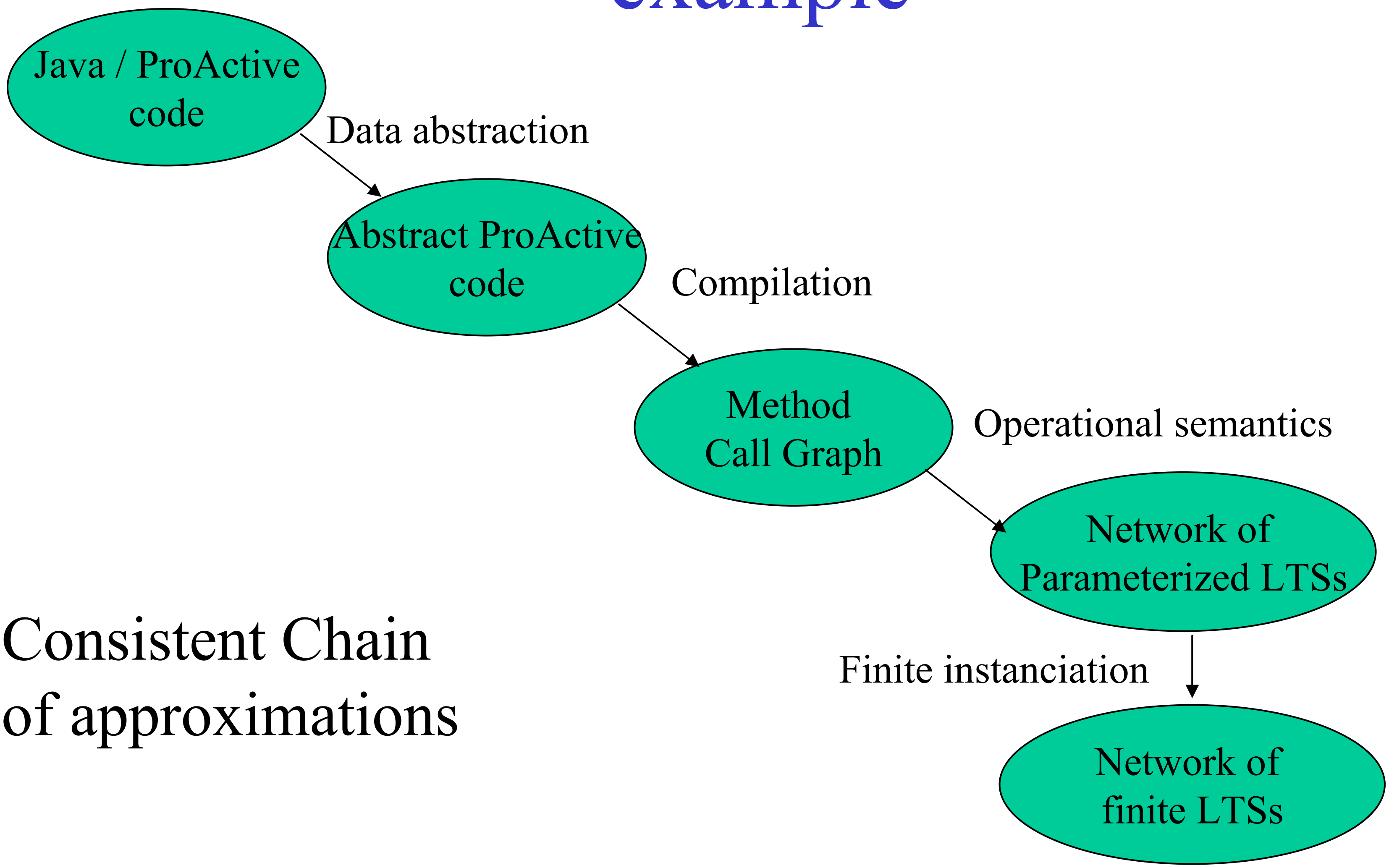
- **Galois connections :**
  - a pair of functions (α,γ) such that:

$$\mathbf{L}^{\#}, \subseteq^{\#} \quad \underset{\alpha}{\overset{\gamma}{\rightleftarrows}} \quad \mathbf{L}^{\mathbf{b}}, \subseteq^{\mathbf{b}}$$

(abstract) (concrete)

- where :
- $\subseteq^{\#}$ and $\subseteq^{\mathbf{b}}$ are information orders
- α and γ are monotonous
- $\alpha(v^{\mathbf{b}}) \subseteq^{\#} v^{\#} \iff v^{\mathbf{b}} \subseteq^{\mathbf{b}} \gamma(v^{\#})$

Abstract Interpretation (6) example



Consistent Chain of approximations

Abstract Interpretation

Summary:

- From Infinite to Finite / Decidable

– library of abstractions for mathematical objects
– information loss : chose the right level !
– composition of abstractions
– sound abstractions :
  property true on the abstract model => true on concrete model
– but incomplete :
  abstract property false => concrete property may be true

Ref: *Abstract interpretation-based formal methods and future challenges,*
P. Cousot, in "informatics 10 years back, 10 years ahead", LNCS 2000.

Program of the course: 1: Semantic Formalisms

- Semantics and formal methods:
  - motivations, definitions, examples
- Denotational semantics : give a precise meaning to programs
  - abstract interpretation
- Operational semantics, behaviour models : represent the complete behaviour of the system
  - CCS, Labelled Transition Systems

Operational Semantics (Plotkin 1981)

- Describes the computation
- States and configuration of an abstract machine:
  - Stack, memory state, registers, heap...
- Abstract machine transformation steps
- Transitions: current state -> next state

Several different operational semantics

Natural Semantics : big steps (Kahn 1986)

- **Defines the results of evaluation.**
- **Direct relation from programs to results**

env |- prog => result

  - env: binds variables to values
  - result: value given by the execution of prog

Reduction Semantics : small steps

describes **each elementary step** of the evaluation

- **rewriting relation** : reduction of program terms
- **stepwise reduction**: <prog, s> -> <prog', s '>
  - infinitely, or until reaching a normal form.

Differences: small / big steps

- Big steps:
  - abnormal execution : add an « error » result
  - non-terminating execution : problem
    - deadlock (no rule applies, evaluation failure)
    - looping program (infinite derivation)
- Small steps:
  - explicit encoding of non termination, divergence
  - confluence, transitive closure ->*

Natural semantics: examples (big steps)

- **Type checking** :

Terms: X | tt | ff | not t | n | t1 + t2 | if b then t1 else t2

Types: Bool, Int

- **Judgements :**

Typing: Γ |- P : τ

Reduction: Γ |- P $\Rightarrow$ v

Deduction rules

Values and expressions:

$$\Gamma \mid\!- \ tt : Bool \qquad\qquad \Gamma \mid\!- \ tt \Rightarrow true$$

$$\Gamma \mid\!- \ ff : Bool \qquad\qquad \Gamma \mid\!- \ ff \Rightarrow false$$

$$\frac{\Gamma \mid\!- \ t1 : Int \qquad \Gamma \mid\!- \ t2 : Int}{\Gamma \mid\!- \ t1 + t2 : Int} \qquad\qquad \frac{\Gamma \mid\!- \ t1 \Rightarrow n1 \qquad \Gamma \mid\!- \ t2 \Rightarrow n2}{\Gamma \mid\!- \ t1 + t2 \Rightarrow n1+n2}$$

Deduction rules

- Environment :

$\delta :: \{x\text{->}v\} \mid\!\!- \ x \Rightarrow v$ $\qquad\qquad$ $\delta :: \{x : \tau\} \mid\!\!- \ x : \tau$

- Conditional :

$$\frac{\Gamma \mid\!\!- \ b \Rightarrow \text{true} \qquad \Gamma \mid\!\!- \ e1 \Rightarrow v}{\Gamma \mid\!\!- \ \text{if } b \text{ then } e1 \text{ else } e2 \Rightarrow v}$$

Exercice : typing rule ?

Operational semantics: big steps for reactive systems **Behaviours**

- **Distributed, synchronous/asynchronous programs:**
  transitions represent communication events
- **Non terminating systems**
- **Application domains:**
  - telecommunication protocols
  - reactive systems
  - internet (client/server, distributed agents, grid, e-commerce)
  - mobile / pervasive computing

Synchronous and asynchronous languages

- Systems build from communicating componants : parallelism, communication, concurrency
- Asynchronous Processes
  - Synchronous communications (rendez-vous)
    Process calculi: CCS, CSP, Lotos
  - Asynchronous communications (message queues)
    SDL modelisation of channels
- Synchronous Processes (instantaneous diffusion)
  Esterel, Sync/State-Charts, Lustre

Exercice: how do you classify ProActive ?

CCS

(R. Milner, “A Calculus of Communicating Systems”, 1980)

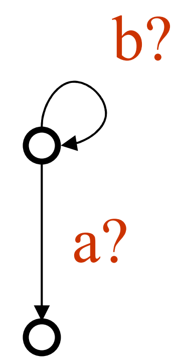
- Parallel processes communicating by Rendez-vous :

a?:b!:nil $\xrightarrow{a?}$ **b!:nil** $\xrightarrow{b!}$ **nil**

a?:P || a!:Q $\xrightarrow{\tau}$ **P || Q**

- Recursive definitions :

let rec { st0 = a?:st1 + b?:st0 } in st0



CCS : behavioural semantics (1)

nil (or skip)

$$\mathbf{a}{:}P \xrightarrow{\mathbf{a}} P$$

$$\frac{P \xrightarrow{\mathbf{a}} P'}{P+Q \xrightarrow{\mathbf{a}} P'} \qquad \frac{Q \xrightarrow{\mathbf{a}} Q'}{P+Q \xrightarrow{\mathbf{a}} Q'}$$

CCS : behavioural semantics (2)

Emissions & réceptions are dual actions

τ invisible action (internal communication)

$$\frac{P \xrightarrow{a} P'}{P||Q \xrightarrow{a} P'||Q} \qquad \frac{Q \xrightarrow{a} Q'}{P||Q \xrightarrow{a} P||Q'}$$

$$\frac{P \xrightarrow{a!} P' \qquad Q \xrightarrow{a?} Q'}{P||Q \xrightarrow{\tau} P'||Q'}$$

$$\frac{[\mu X.P/X]P \xrightarrow{a} P'}{\mu X.P \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{a} P' \quad \mathbf{a} \notin \{\mathbf{b?},\mathbf{b!}\}}{\mathit{local}\ \mathbf{b}\ \mathit{in}\ P \xrightarrow{a} \mathit{local}\ \mathbf{b}\ \mathit{in}\ P'}$$

Derivations (construction of each transition step)

$$\frac{}{a?{:}P \xrightarrow{a?} P}\ \text{Prefix}$$

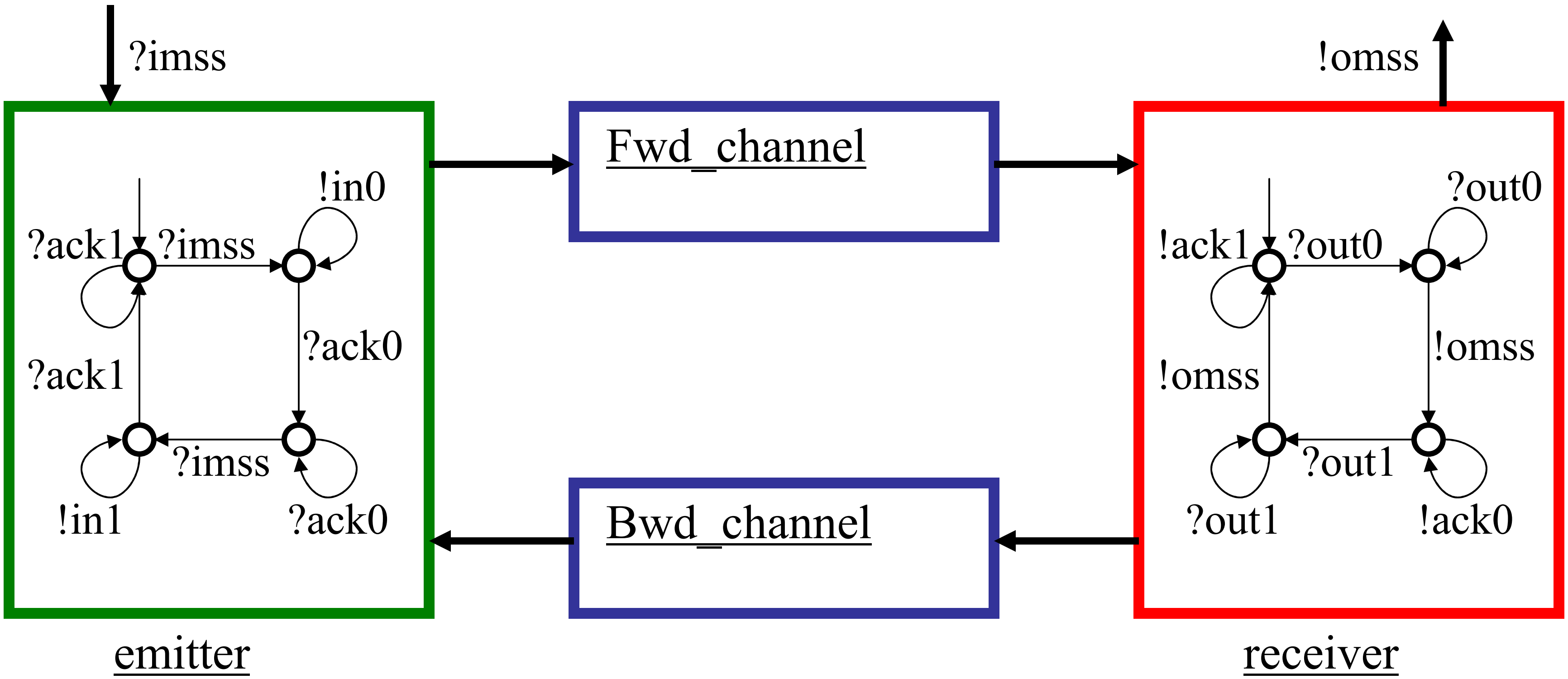
$$\frac{}{a?{:}P \,||\, Q \xrightarrow{a?} P \,||\, Q}\ \text{Par-L} \qquad \frac{}{a!{:}R \xrightarrow{a!} R}\ \text{Prefix}$$

$$\frac{}{(a?{:}P \,||\, Q) \,||\, a!{:}R \xrightarrow{\tau} (P \,||\, Q) \,||\, R}\ \text{Par-2}$$

$$(a?{:}P \,||\, Q) \,||\, a!{:}R \xrightarrow{a?} (P \,||\, Q) \,||\, a!{:}R$$

Par-L(Par_L(Prefix))

Example: Alternated Bit Protocol



Write in CCS ?

Hypotheses: channels can loose messages

Requirement:

the protocol ensures no loss of messages

Example: Alternated Bit Protocol (2)

- **emitter** =

  let rec {em0 = ack1? :em0 + imss?:em1

  and em1 = in0! :em1 + ack0? :em2

  and em2 = ack0? :em2 + imss? :em3

  and em3 = in1! :em3 + ack1? :em0

  }

  in em0

- **ABP =** local {in0, in1, out0, out1, ack0, ack1, …}

  in emitter || Fwd_channel || Bwd_channel || receiver

Example: Alternated Bit Protocol (3)

Channels that loose and duplicate messages (in0 and in1) but preserve their order ?

- Exercise :
  1) Draw an automaton describing the loosy channel behaviour
  2) Write the same description in CCS

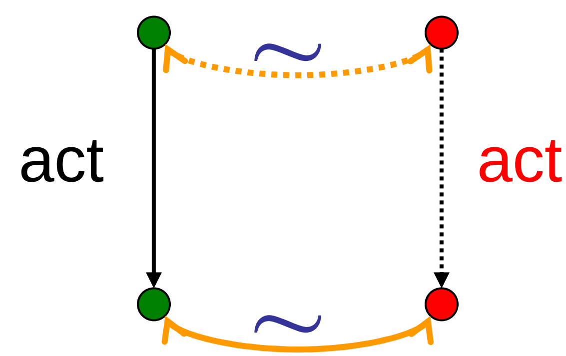
Bisimulation

- **Behavioural Equivalence**
  - non distinguishable states by observation:

    two states are equivalent if for all possible action, there exist equivalent resulting states.

- **minimal automata**

quotients = canonical normal forms



Some definitions

- **Labelled Transition System (LTS)**

  (S, s0, L, T)

  where: S is a set of states

  $s0 \in S$ is the initial state

  L is a set of labels

  $T \subseteq SxLxS$ is the transition relation

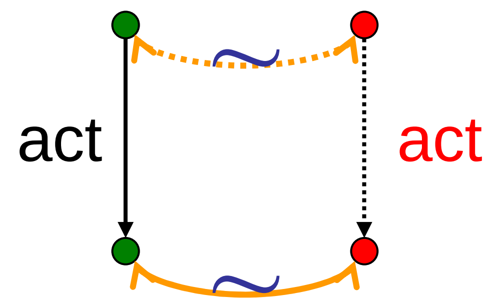
- **Bisimulations**

  $R \subseteq SxS$ is a bisimulation iff
  - It is a equivalence relation
  - $\forall (p,q) \in R$,

    $(p,l,p') \in T \Rightarrow \exists q' / (q,l,q') \in T$ and $(p',q') \in R$

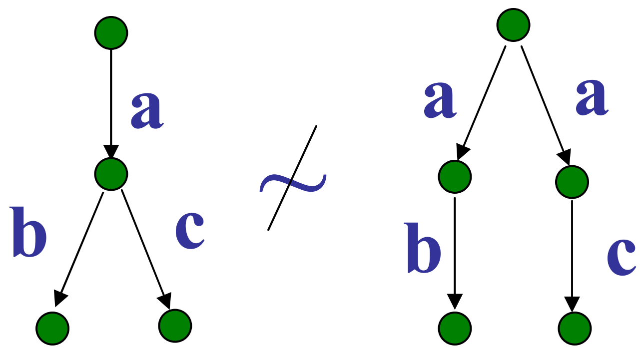
  ~ is the coarsest bisimulation

  2 LTS are bisimilar iff their initial states are in ~



Bisimulation (3)

- More precise than trace equivalence :



- Congruence for CCS operators :

for any CCS context C[.], C[P] ~ C[Q] <=> P~Q

Basis for compositional proof methods

Bisimulation (4)

- Congruence laws:

  P1~P2 => a:P1 ~ a:P2 (∀ P1,P2,a)

  P1~P2, Q1~Q2 => P1+Q1 ~ P2+Q2

  P1~P2, Q1~Q2 => P1||Q1 ~ P2||Q2

  Etc…

Bisimulation : Exercice

Next courses

2) Application to distributed applications

- ProActive : behaviour models
- Tools : build an analysis platform

3) Distributed Components

- Fractive : main concepts
- Black-box reasoning
- Deployment, management, transformations

www-sop.inria.fr/oasis/Eric.Madelaine

⇨ **Teaching**