CentraleSupélec

Sémantique et vérification

Frédéric Boulanger (avec le concours de Cécile Hardebolle)

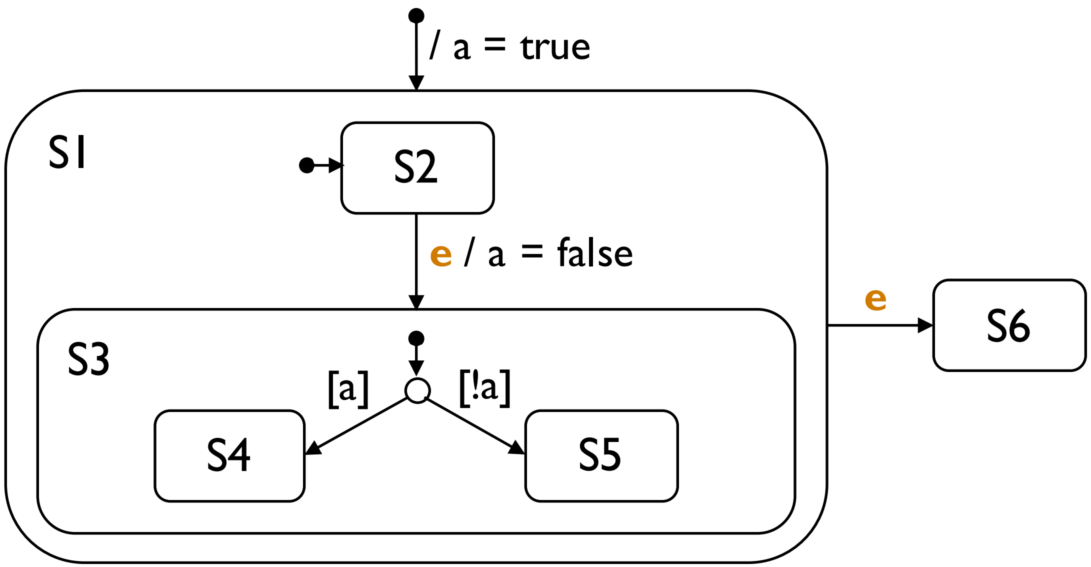
What is semantics?

- What is the meaning of jaguar?

The problem with semantics…

Taken from:
"UML vs. Classical vs. Rhapsody Statecharts: Not All Models are Created Equal"
Michelle Crane, Juergen Dingel

- What is the behavior described by this Statechart diagram when the event e occurs?



- Event e may lead to:
  - S4 with UML: outer transition to S1 has priority and sets a to true
  - S5 with Rhapsody: transition from S2 to S3 has priority and sets a to false
  - S6 with Stateflow: outer transition preempts state S1

Explicit definition of semantics

- All three meanings for the diagram are correct…
  …The problem is that the semantics is implicitly defined by the tool !

  - What if:
    - The designer of a system thinks according to UML semantics
    - The code generator interprets the model according to Rhapsody's semantics
    - The verification is made according to Stateflow's semantics

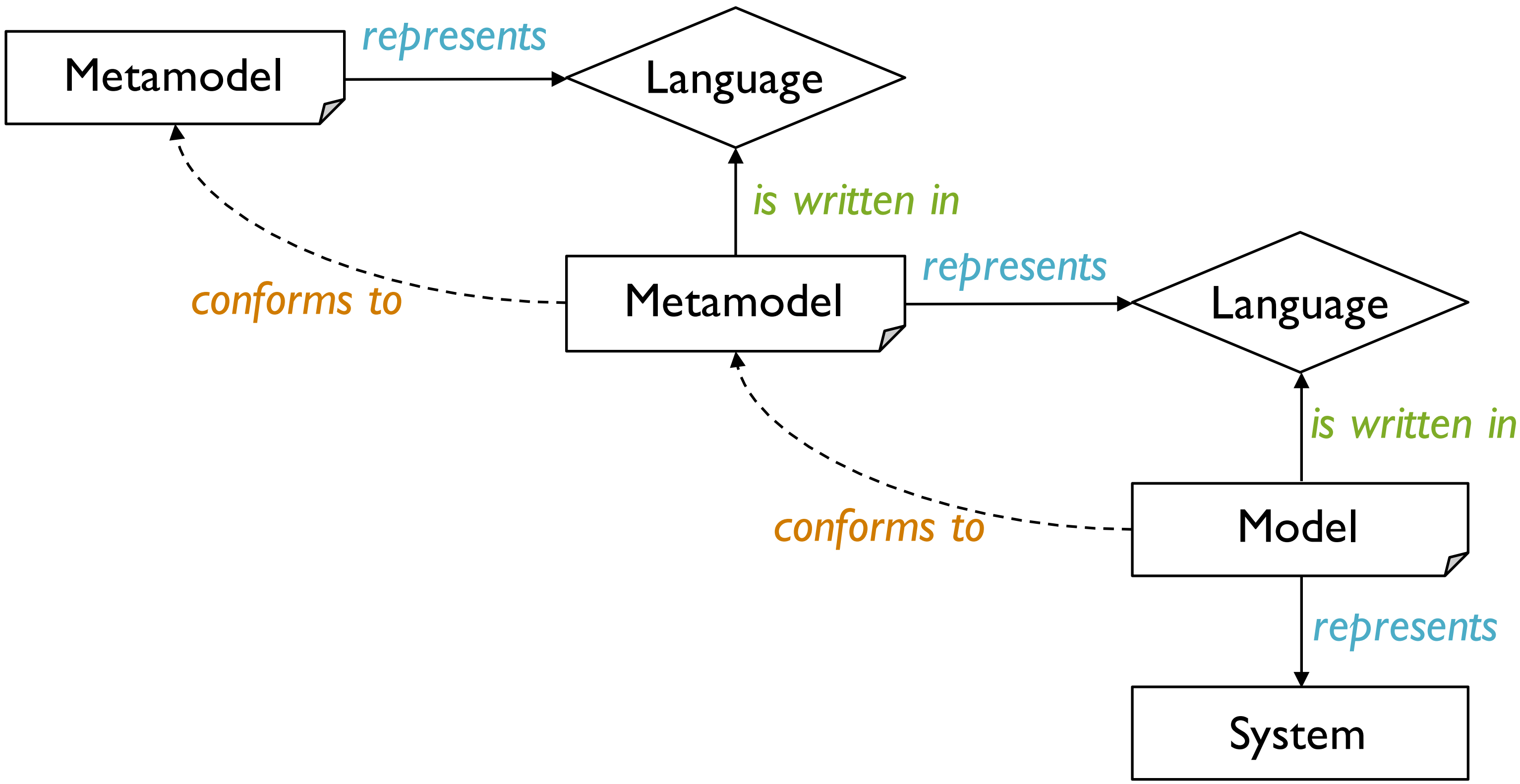
    } ?

☛ The semantics of a model should be:

- Explicit, so that there is no doubt about how to interpret it
- Well defined, so that the properties of the model can be verified

- Formal semantics = semantics defined in such a way that a model can be processed automatically in a consistent way by programs

Model, metamodel and modeling language



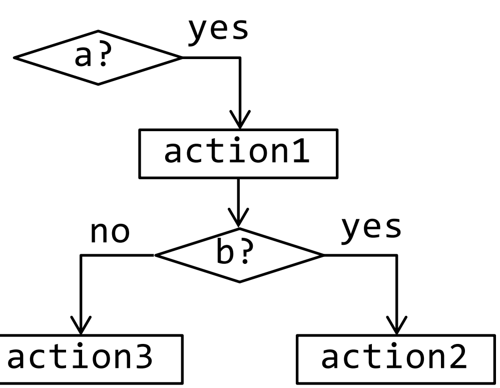
Defining the semantics of a language

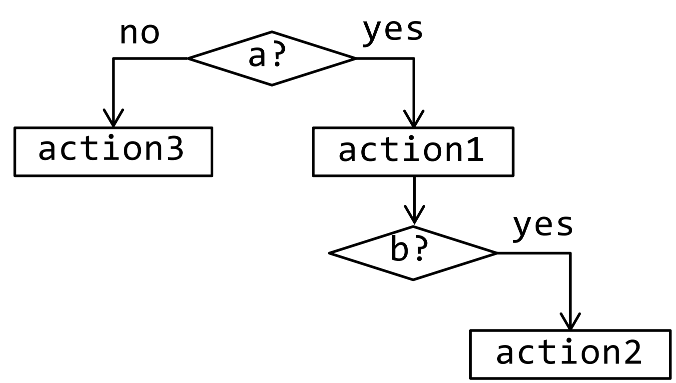
- The formal semantics of a language is based on its syntax
  - Abstract syntax = concepts and relations (metamodel)
  - Concrete syntaxes = text or graphics that obey a grammar

"Things must be *well written* to be *well understood*"

```
if (a) then
do action1
if (b) then
do action2
else
do action3
```

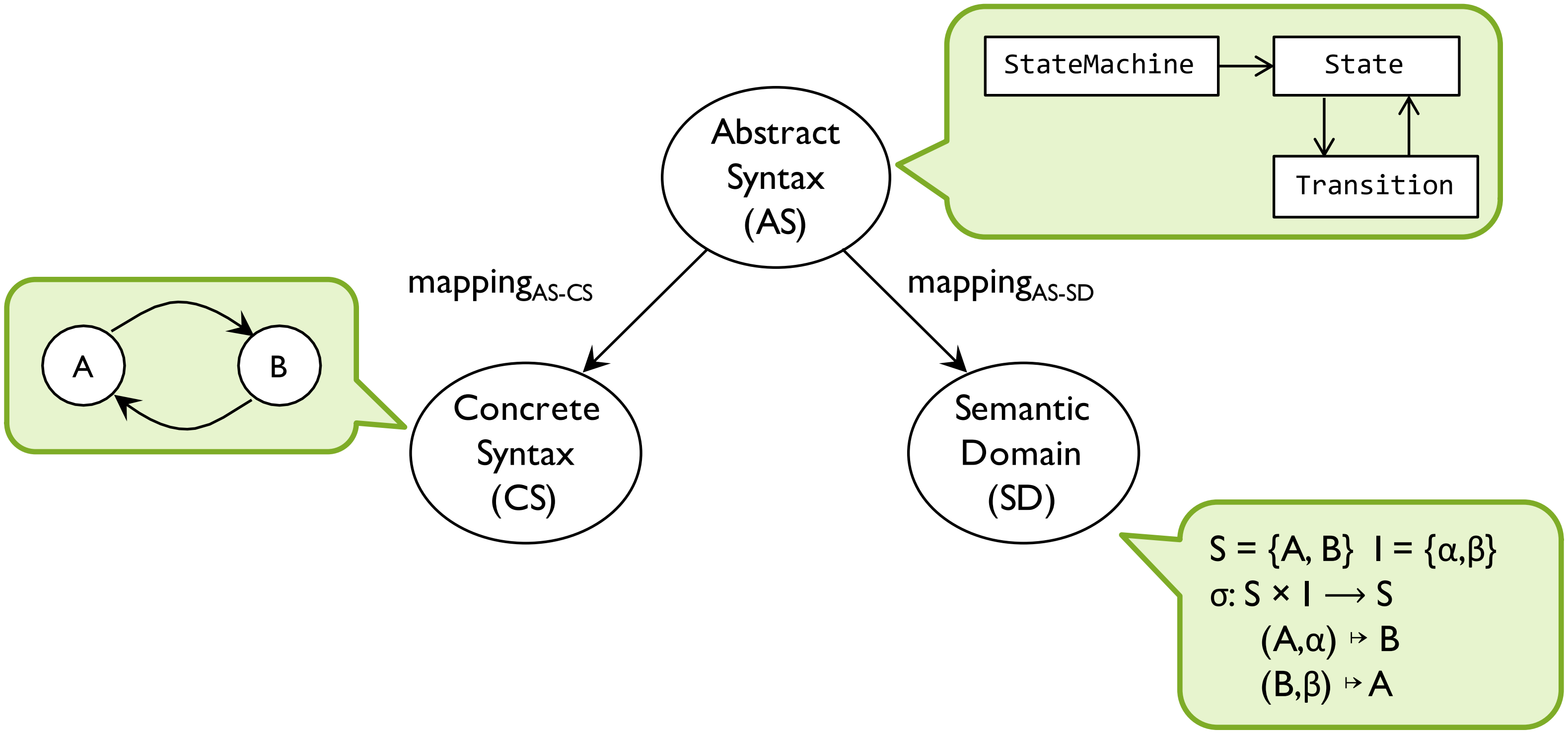
Does the `else` correspond to:
- not a?
- a and not b?





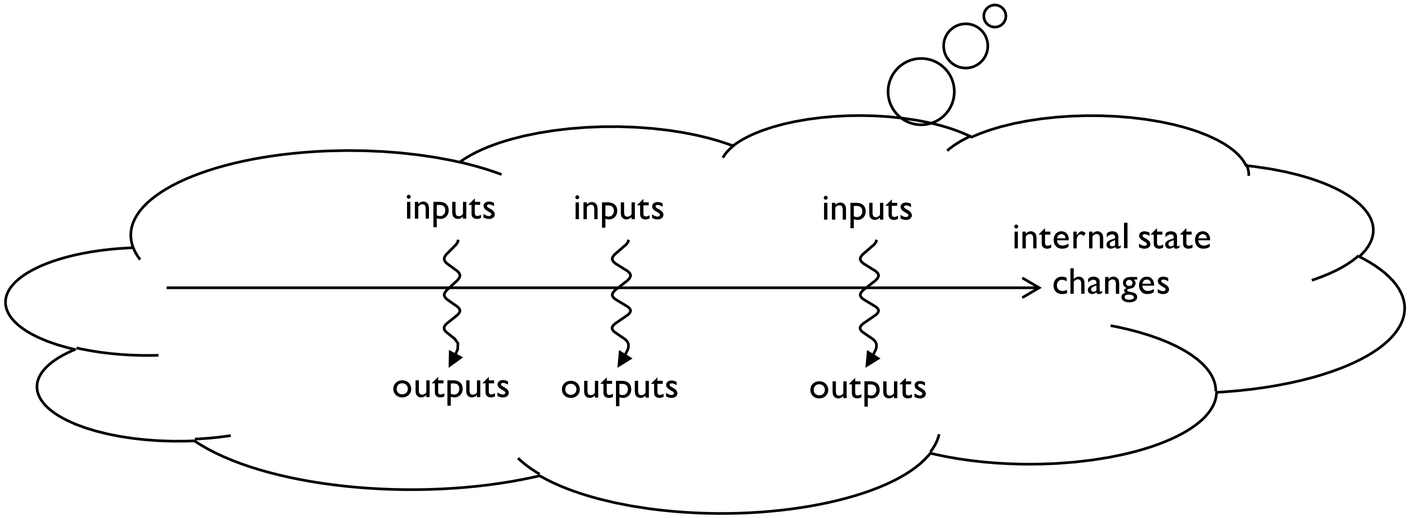
Defining the semantics of a language

- How to define the semantics?
  1. Choose a semantic domain (other language or mathematics)
  2. Define a mapping of the syntactic elements to items in the semantic domain



Execution semantics

How to describe the execution of a model?



☛ Semantic domain = abstract execution machine

Abstract execution machine = state + primitive operations

➥ The execution of the model is described in terms of changes in the state of the machine

Different flavors of semantics

- Operational semantics describes the execution of a model as a series of state changes of the execution machine
  - Example: how to swap two integers a and b?

Execution machine = state + primitive operations

```
tmp = a;
a = b;
b = tmp
```

`tmp = a` → `a = b` → `b = tmp`

a	*a0*
b	*b0*
tmp	*tmp0*

a	*a0*
b	*b0*
tmp	*a0*

a	*b0*
b	*b0*
tmp	*a0*

a	*b0*
b	*a0*
tmp	*a0*

- Operational semantics describes the complete sequence of states
  ➥ *May be too much detailed…*
  - Example: for the swap behavior, we don't care which variable is overwritten first!

Different flavors of semantics

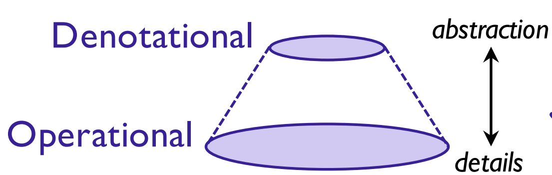
- Denotational semantics describes the path from initial to final state
  - Example: how to swap two integers a and b?

`swap: initial state ⟼ new state`

`swap(a,b)`

a	*a0*
b	*b0*
tmp	*tmp0*

a	*b0*
b	*a0*
tmp	*a0*



2 models
with equivalent *operational* semantics
have equivalent *denotational* semantics

- Denotational semantics describes the change of the complete state
  ➡ *May be too much detailed…*
  - Example: for the swap behavior, we don't care about the value of tmp at the end

Different flavors of semantics

- Axiomatic semantics describes the change of the properties of the state
  - Example: how to swap two integers a and b?

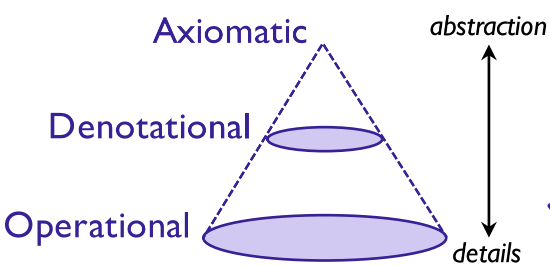
{ a = a0 ∧ b = b0 } `swap(a,b)` { a = b0 ∧ b = a0 }

`swap(a,b)`

a	*a0*
b	*b0*

→

a	*b0*
b	*a0*



2 models
with equivalent *denotational* semantics
have equivalent *axiomatic* semantics

Different semantics, different uses

Formal semantics allows for unambiguous interpretation of models
☛ Execution, verification, computation of properties (timing, power…)

- Operational semantics describes the details of the execution
  - OK for simulation and code generation
  - Example: describe the execution steps for swapping a and b
- Denotational semantics describes the results of the execution
  - OK for verifying the correctness of the results
  - Example: obtain the values of `tmp`, a and b from the initial values of a and b
- Axiomatic semantics describes properties of the execution state
  - OK for verifying invariants, safety properties
  - Example: assert that the values of a and b have been swapped

Semantics and verification

- Well defined semantics $\Rightarrow$ well defined behavior and properties
- Formal semantics $\Rightarrow$ behavior can be analyzed automatically

- Verification is used to check for:
  - Unreachable states (dead code)
  - Properties that should always hold (security)
  - States that should always be reachable (liveness)
  - Forbidden operations (divide by zero, square root of negative number)
  - Value overflow

- Three flavors of verification:
  - Model-checking: complete, automatic, but combinatory explosion
  - Proof: complete, partially automated
  - Test: incomplete

Workflow

① Exploratory "informal" design

- Create a model
- Execute the model (simulate the behavior of the system)
- Iterate until the model seems to behave properly

② Formal design

- Formalize properties from the specification
- Check the properties
  - Properties OK → done
  - Property does not hold → understand why (counter example) and fix it

③ Implementation verification

- Generate code from the model
- Perform static analysis on the code to check that the properties hold
- Generate test scenarios and evaluate their coverage
- Test the real system using the test scenarios

Semantics and verification

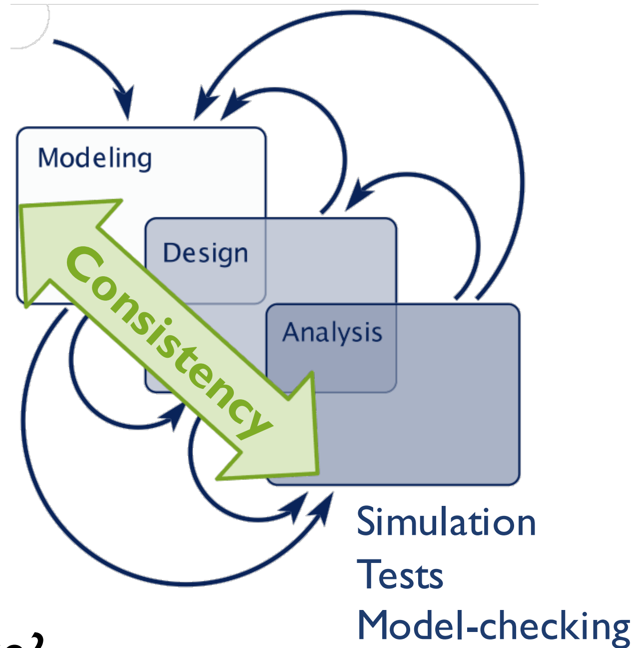
Verification requires:

A. Precise semantics for each model
B. Precise semantics for the interactions between models

A. Tools for the verification of homogeneous models
- SCADE (Esterel Technologies): model-checking of synchronous reactive models
- Simulink Design Verifier (The MathWorks): proofs on Matlab/Simulink models
- Polyspace (The MathWorks): static analysis of C/C++ or Ada code
- Frama C (CEA, INRIA): static analysis of C code
- Krakatoa (Univ. Paris-Sud): static analysis of Java code
- Why (Univ. Paris-Sud): pivot formal language for pre/post semantics
- … and many other theorem provers

Some issues with verification…

- Is the proof you made on the model of the system really valid on the system?
  - ➡ What You Prove Is What You Execute (WYPIWYE)



- Did you really prove what you wanted to prove?
  - ➡ What You Prove Is What You Mean (WYPIWYM)

$\square((up \wedge \neg obstacle) \Rightarrow \lozenge\ power = 1) \wedge (down \Rightarrow \lozenge\ power = -1))$

"When the user puts the switch in the up position the window closes unless there is an obstacle, and when the user puts the switch in the down position the window opens." (*liveness*)