Modélisation du comportement des systèmes 2

Frédéric Boulanger

Novembre 2019

Rappels

Modèle représentation simplifiée (abstraction) d'un système pour un objectif donné

- modèle d'analyse (système pré-existant)
- modèle de conception (système à concevoir)
- modèle explicatif
- modèle prédictif
- modèle exécutable/constructif

Système objet d'étude, caractérisé par une structure et des interactions entre ses composants

Conception processus composé d'une suite de raffinements de modèles depuis la spécification jusqu'à l'implementation

MDE/IDM approche de la conception dans laquelle les modèles capturent la majeure partie de l'effort de conception

Rappels

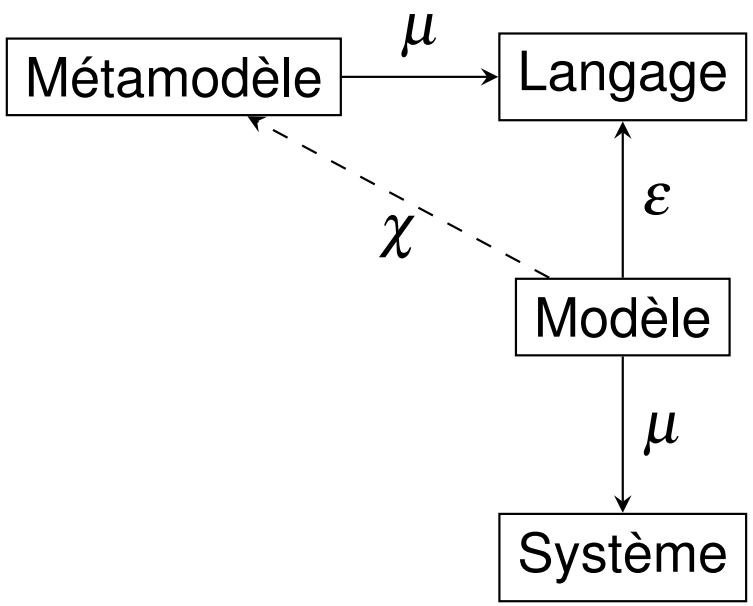
Relations entre modèles

$M \xrightarrow{\mu} S$ M représente (est un modèle de) S

$M \xrightarrow{\varepsilon} L$ M appartient au (est écrit dans le) langage L

$M \xrightarrow{\delta} P$ M se décompose en (contient) P

$M \xrightarrow{\chi} MM$ M est conforme au métamodèle MM



Sémantique des modèles

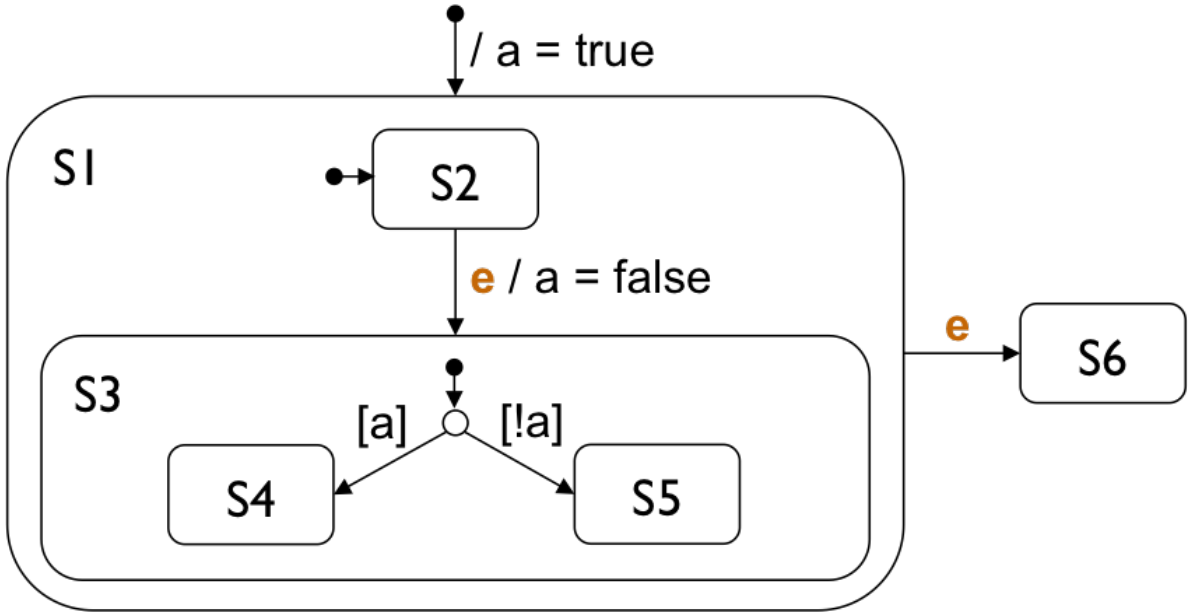
Sémantique des modèles

Sémantique des modèles

Quel est le sens de « jaguar » ?

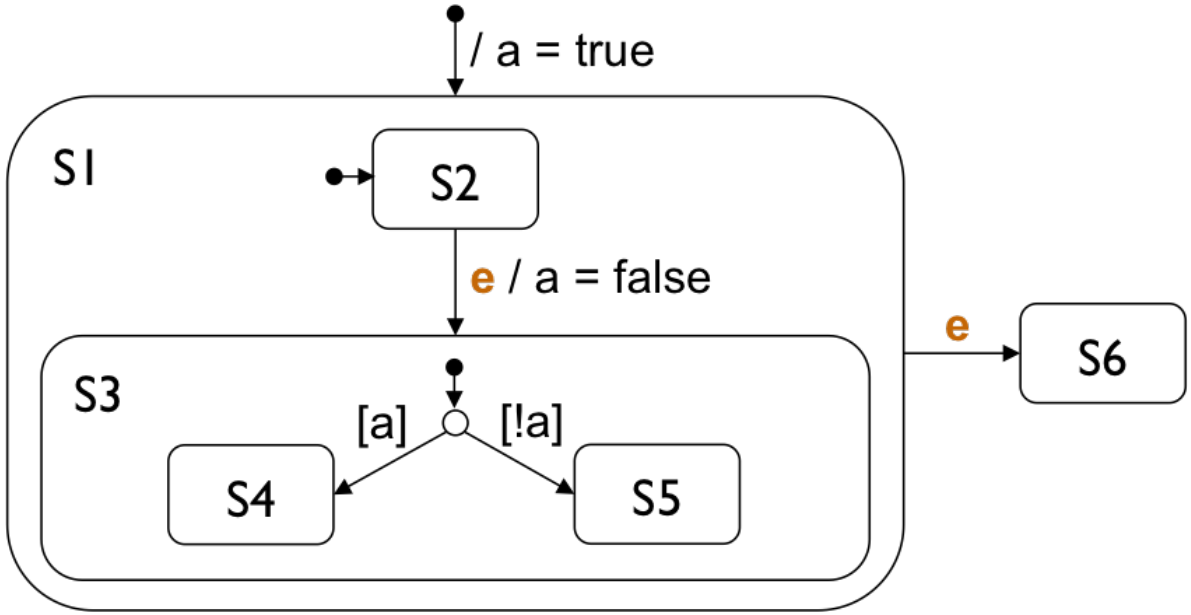
Nécessité d'une sémantique précise

Un diagramme Statechart



Nécessité d'une sémantique précise

Un diagramme Statechart



L'évènement `e` peut mener à :

- S4 avec UML : la transition vers S1 a priorité et met `a` à `true`
- S5 avec Rhapsody : la transition de S2 vers S3 a priorité
- S6 avec Stateflow : la transition vers S6 préempte S1

Nécessité d'une sémantique précise

Les 3 sémantiques sont *correctes*

Le problème est que la sémantique est définie *implicitement* par l'outil !
Que se passe-t-il si :

- le *concepteur* pense selon la sémantique UML
- le générateur de code interprète le modèle comme Rhapsody
- la vérification est faite selon la sémantique Stateflow ?

Nécessité d'une sémantique précise

Les 3 sémantiques sont *correctes*

Le problème est que la sémantique est définie *implicitement* par l'outil !
Que se passe-t-il si :

- le *concepteur* pense selon la sémantique UML
- le générateur de code interprète le modèle comme Rhapsody
- la vérification est faite selon la sémantique Stateflow ?

La sémantique d'un modèle devrait être :

- *explicite*, pour éliminer les doutes sur l'interprétation
- *bien définie*, de façon à pouvoir vérifier des propriétés du modèle

Nécessité d'une sémantique précise

Les 3 sémantiques sont *correctes*

Le problème est que la sémantique est définie *implicitement* par l'outil !
Que se passe-t-il si :

- le *concepteur* pense selon la sémantique UML
- le générateur de code interprète le modèle comme Rhapsody
- la vérification est faite selon la sémantique Stateflow ?

La sémantique d'un modèle devrait être :

- *explicite*, pour éliminer les doutes sur l'interprétation
- *bien définie*, de façon à pouvoir vérifier des propriétés du modèle

Sémantique formelle

Sémantique définie de telle manière qu'un modèle puisse être traité de façon automatique et consistante par des programmes.

Syntaxe et sémantique

Modèle informatique

- un modèle est considéré en tant que système (transformation, validation)
- un modèle est écrit dans un langage de modélisation (UML, Simulink, ...)
- un langage de modélisation est considéré en tant que système (éditeurs, grammarware)
- un langage de modélisation est décrit par un métamodèle (modèle du langage)

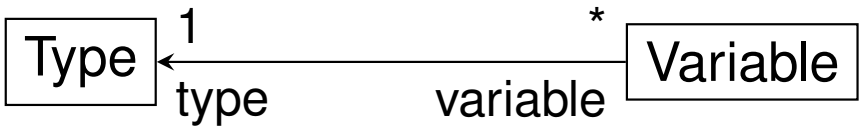
Métamodèle = syntaxe seule

- syntaxe abstraite (concepts et relations)
- syntaxe concrète (mots-clefs, éléments graphiques)

Syntaxes abstraite et concrète

Exemple

- syntaxe abstraite (concepts et relations)



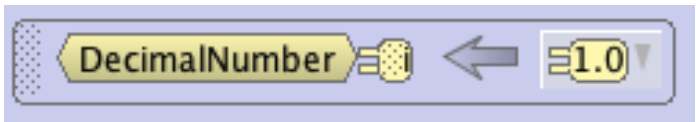
- syntaxe concrète (notation à l'aide de symboles)

```
int i;
```

- autre syntaxe concrète textuelle

```
i: int;
```

- syntaxe concrète graphique

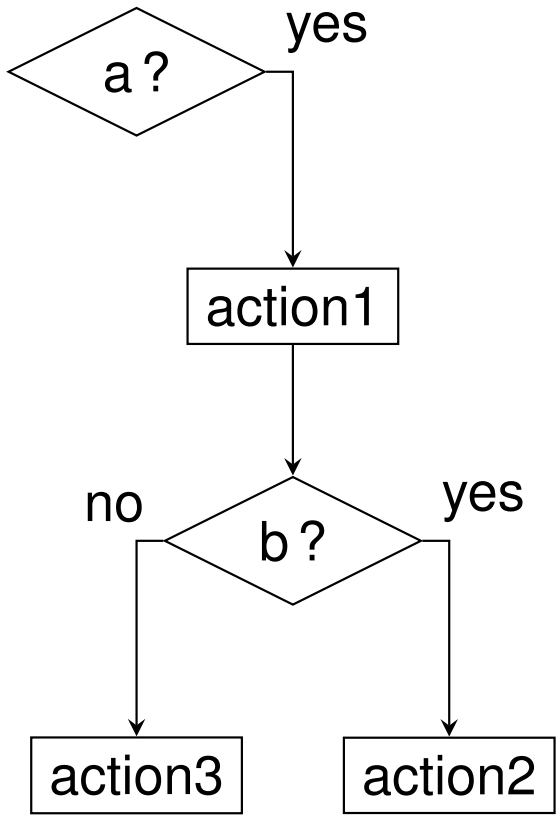


Nécessité d'une syntaxe non ambigüe

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

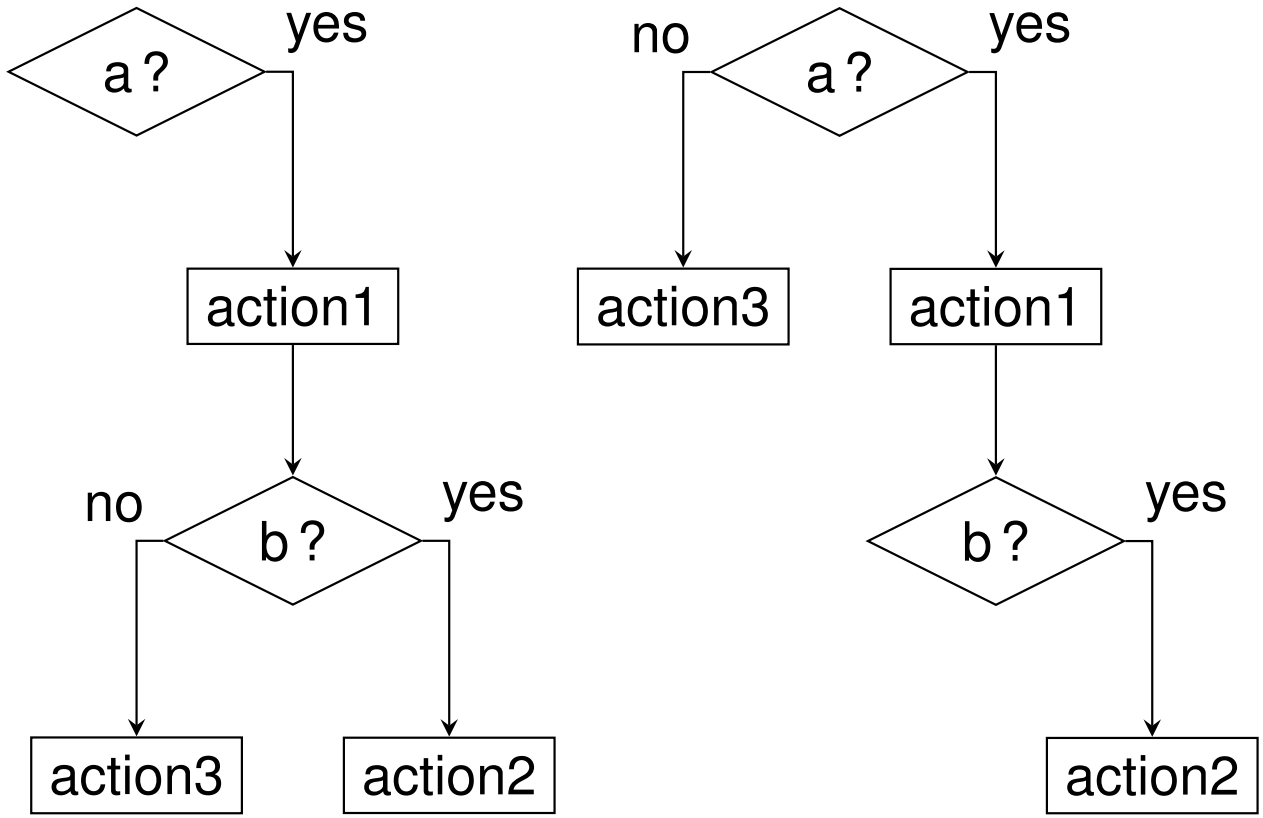
Nécessité d'une syntaxe non ambigüe

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



Nécessité d'une syntaxe non ambigüe

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



Sémantique

Sémantique : sens donné aux éléments syntaxiques
Se définit par une association entre les éléments syntaxiques (signifiants) et les éléments du domaine sémantique (signifiés)

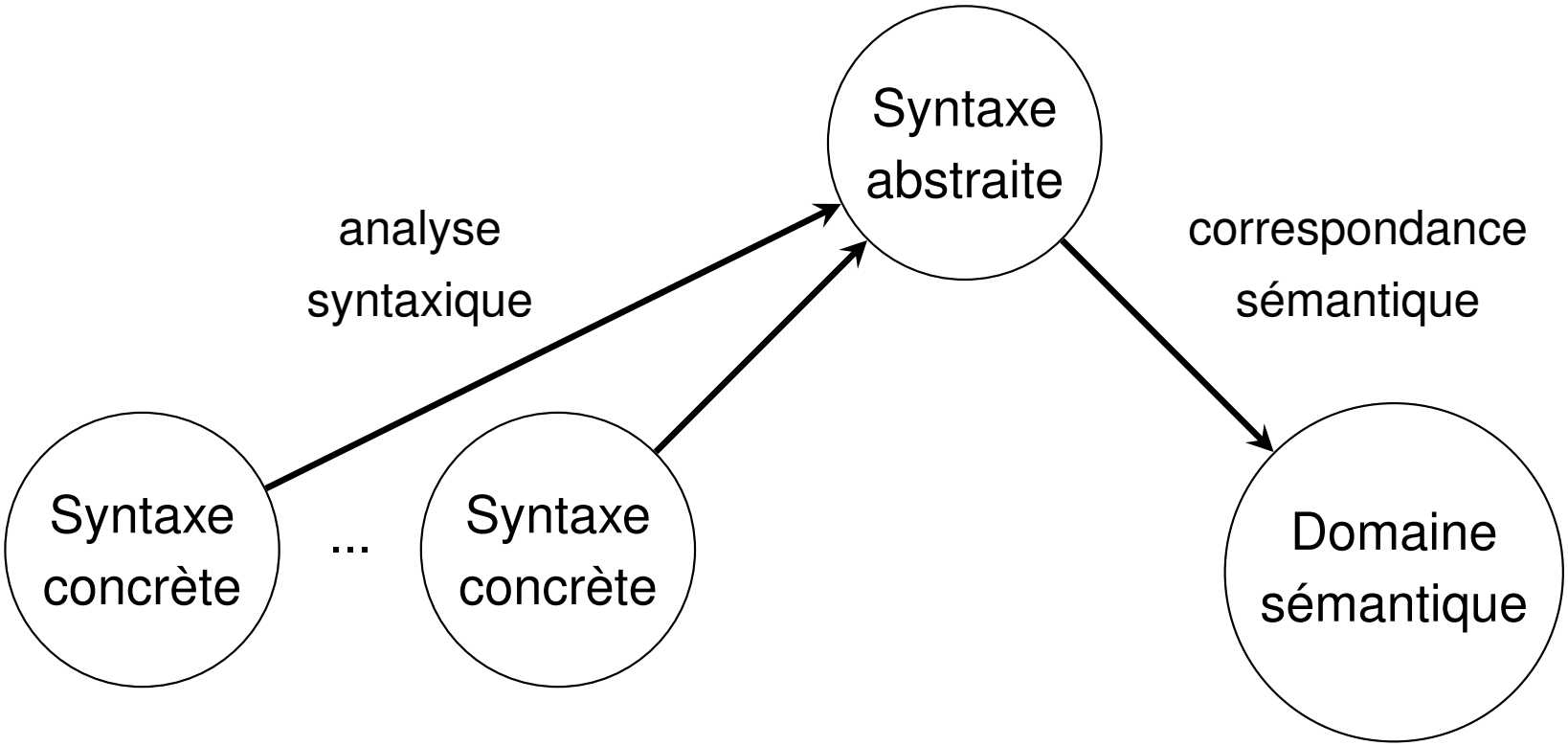
Domaine sémantique

- autre langage (traduction)
- théorie mathématique

Définition de la sémantique

1. définition formelle de la syntaxe abstraite
2. choix d'un domaine sémantique
3. association entre syntaxe abstraite et domaine sémantique

Définition de la sémantique d'un language



Sémantique opérationnelle

Définit la sémantique d'un langage en décrivant l'effet des éléments syntaxiques sur une machine d'exécution.

Exemple

Machine d'exécution = mémoire de mots de 32 bits + table d'association nom/adresse + addition modulo 2^{32}

- `int i = 0;` la machine réserve un mot,
  met ses bits à `0x0000 0000`
  et associe son adresse au symbole `i`
- `int j = 1;` la machine réserve un mot,
  met ses bits à `0x0000 0001`
  et associe son adresse au symbole `j`
- `i = i + j;` la machine additionne modulo 2^{32} les mots
  associés aux symboles `i` et `j` et range
  le résultat à l'adresse associée au symbole `i`

Sémantique opérationnelle — Exemple

Syntaxe

- *declaration* ::= `int` *varname* = *value* **`;`** ;;
- *assignment* ::= *varname* = *varname* + *varname* **`;`** ;;
- *varname* ::= `a` | `b` | `...` | `z` ;;
- *value* ::= *integer* | −*integer* ;;
- *integer* ::= *digit* | *integer digit* ;;
- *digit* ::= `0` | `1` | `2` | `3` | `4` | `5` | `6` | `7` | `8` | `9` ;;

Sémantique opérationnelle — Exemple

Domaine sémantique

- valeurs entières en complément à 2 sur 32 bits
- arithmétique modulo 2^{32}
- déclarations et affectation : machine d'exécution

a		
b		
c		
...	...	...
i		
j		
...	...	...
z		

Sémantique opérationnelle — Exemple

Association syntaxe – sémantique

- déclaration `int i = 1;`

a		
...	...	...
i	•	0000 0000 0000 0000 0000 0000 0000 0001
...	...	...
z		

- valeurs entières :

  value ::= *integer* | −*integer* ;;

  binaire complément à 2

  integer ::= *digit* | *integer digit* ;;

  valeur ×10 +

- opération + : addition modulo 2^{32}
- opération = : écriture dans le mot

Sémantique statique

Programme syntaxiquement correct

```
int i = 1;
int j = 2;
k = i + j;
```

Sémantique de `k = i + j;` ?

Aucun mot n'est associé au symbole `k`.

Sémantique statique

Contraintes sur la validité d'un modèle
qui ne peuvent pas être capturées par la syntaxe.

Sémantique opérationnelle — limites

Programme P1

```
int i = 1;
int j = 2;
j = i + j;
```

Programme P2

```
int j = 2;
int i = 1;
j = i + j;
```

Inconvénient

Ces deux programmes font passer la machine d'exécution par des suites d'états différentes, ils n'ont donc pas la même sémantique opérationnelle.

Pourtant, ils donnent le même résultat.

Sémantique dénotationnelle

Afin de ne pas dépendre de la succession des états de la machine d'exécution, la sémantique dénotationnelle ne s'intéresse qu'aux états initial et final de la machine.
À chaque programme est associée une fonction mathématique appelée *dénotation*, qui représente l'effet du programme.

- denotation(P) : état avant P $\mapsto$ état après P

La dénotation d'un programme s'obtient par composition des dénotation de ses constituants :

$$\begin{aligned}\delta(\texttt{int i= 1; int j = 2; j = j + i;}) &= \delta(\texttt{j = j + i;}) \circ \delta(\texttt{int j = 2;}) \circ \delta(\texttt{int i = 1;}) \\ &= \bot \mapsto \{(i = 1) \wedge (j = 3)\}\end{aligned}$$

Les étapes intermédiaires de l'exécution n'apparaissent pas dans cette sémantique.
Exemple : Instructions en Isabelle

Sémantique axiomatique

La sémantique dénotationnelle décrit l'évolution de l'état complet de la machine d'exécution.

Programme P3

```
int tmp = i;
i = j + 0;
j = tmp + 0;
```

Programme P4

```
int tmp = j;
j = i + 0;
i = tmp + 0;
```

P3 et P4 n'ont pas la même sémantique dénotationelle car la valeur de `tmp` n'est pas nécessairement la même à la fin de l'exécution de chacun d'eux.
Pourtant, on aimerait pouvoir dire que ces deux programmes sont équivalents en ce qui concerne la permutation des valeurs associées à `i` et `j`.

Sémantique axiomatique

La sémantique axiomatique décrit l'évolution
des propriétés de l'état de la machine d'exécution.

Programme P3

```
int tmp = i;
i = j + 0;
j = tmp + 0;
```

Programme P4

```
int tmp = j;
j = i + 0;
i = tmp + 0;
```

P3 et P4 n'ont pas la même sémantique dénotationelle car la valeur de `tmp` n'est pas nécessairement la même à la fin de l'exécution de chacun d'eux.
Pourtant, on aimerait pouvoir dire que ces deux programmes sont équivalents en ce qui concerne la permutation des valeurs associées à `i` et `j`.

$$\left(\mathtt{i} = i_0 \wedge \mathtt{j} = j_0\right) \overset{P3}{\Rightarrow} \left(\mathtt{i} = j_0 \wedge \mathtt{j} = i_0\right)$$

Sémantique axiomatique

Sémantique pré-post (logique de Hoare) : $\{\phi\}$ P $\{\psi\}$

- ϕ = précondition
- ψ = postcondition
- axiome d'affectation : $\{\phi[\text{expr}/\text{x}]\}$ `x = expr` $\{\phi\}$
- règles de démonstration : (exemple de l'alternative)

$$\begin{cases} \{\phi \wedge C\} \text{ P1 } \{\psi\} \\ \{\phi \wedge \neg C\} \text{ P2 } \{\psi\} \end{cases} \Rightarrow \{\phi\} \texttt{ if (C) P1 else P2 endif } \{\psi\}$$

Exemple : Instructions en Why3

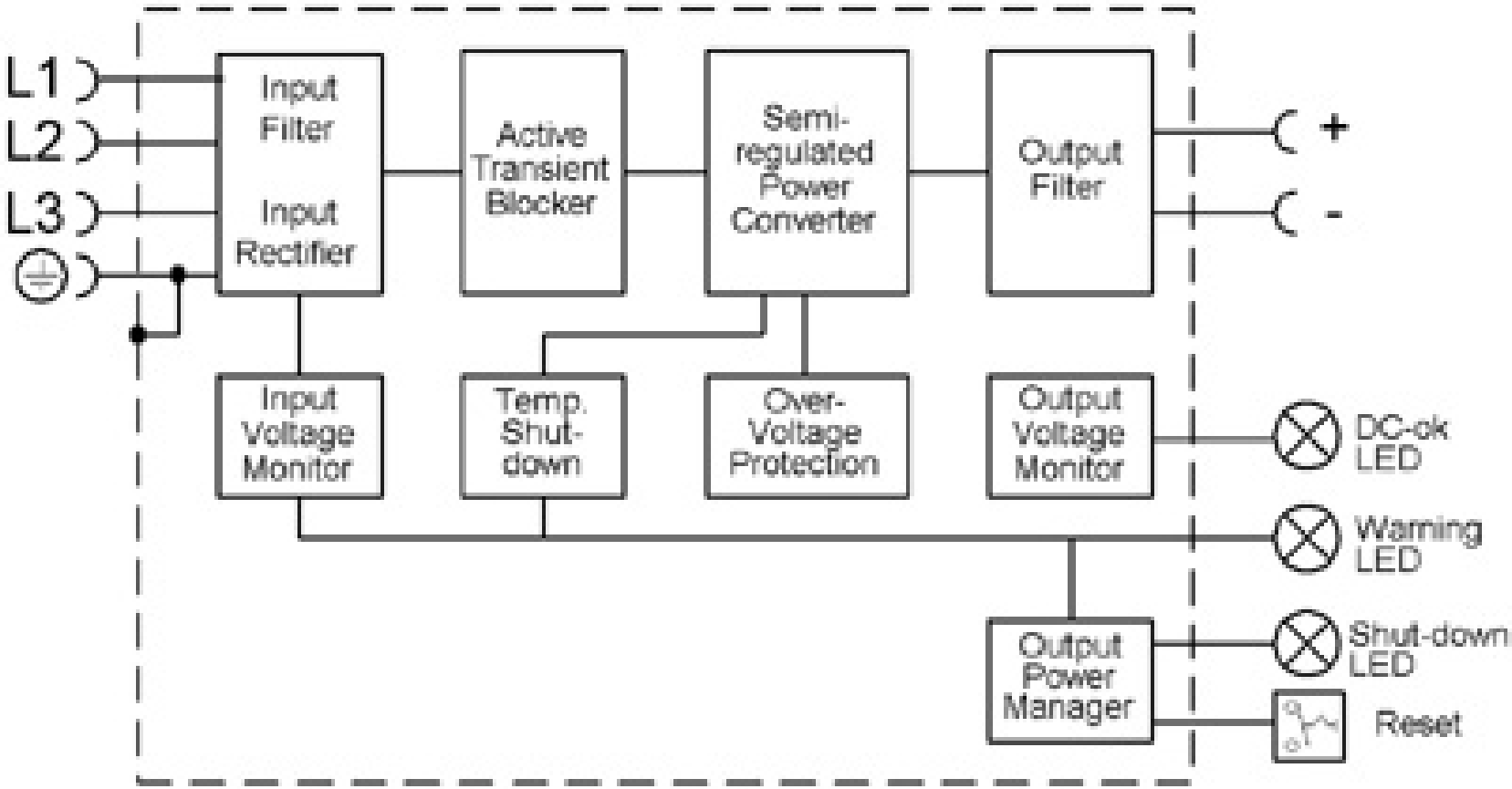
Relations entre les définitions sémantiques

- si P1 et P2 sont syntaxiquement équivalents, ils ont la même sémantique opérationnelle
- si P1 et P2 ont la même sémantique opérationnelle, ils ont la même sémantique dénotationnelle
- si P1 et P2 ont la même sémantique dénotationnelle, ils ont la même sémantique axiomatique
- les réciproques sont fausses

Syntaxes concrètes graphiques

Correspondance sémantique abstraite vers domaine sémantique.
Pas de problème pour les langages graphiques (UML, StateCharts) : on s'appuie sur le métamodèle.

Sémantique abstraite



- blocs (traitements)
- relations entre blocs (communications)

Sémantique abstraite

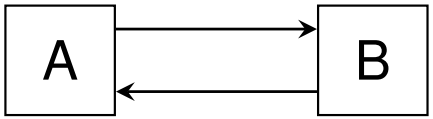
Opérations abstraites

- blocs :
  - `init()`
  - `react()`
  - `finish()`
- ports / relations :
  - `send()`
  - `receive()`
  - `hasData()`

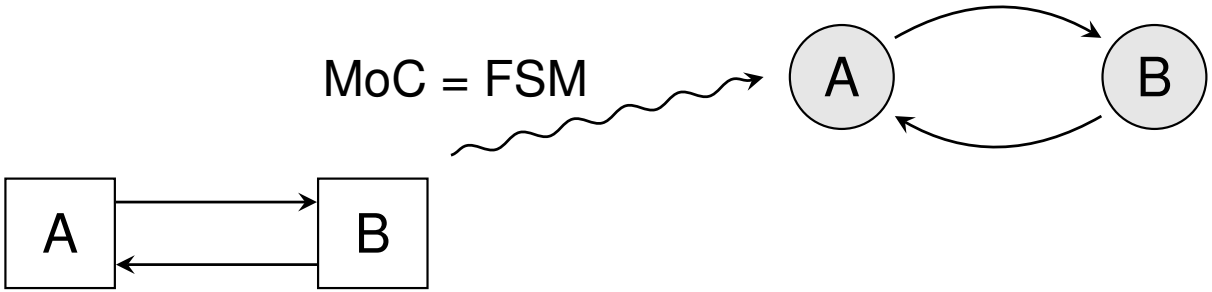
Moteur d'exécution

- utilise les opérations abstraites pour exécuter un modèle
- les opérations sont raffinées en une sémantique concrète
- raffinement = modèle de calcul ou d'exécution

Modèle de calcul

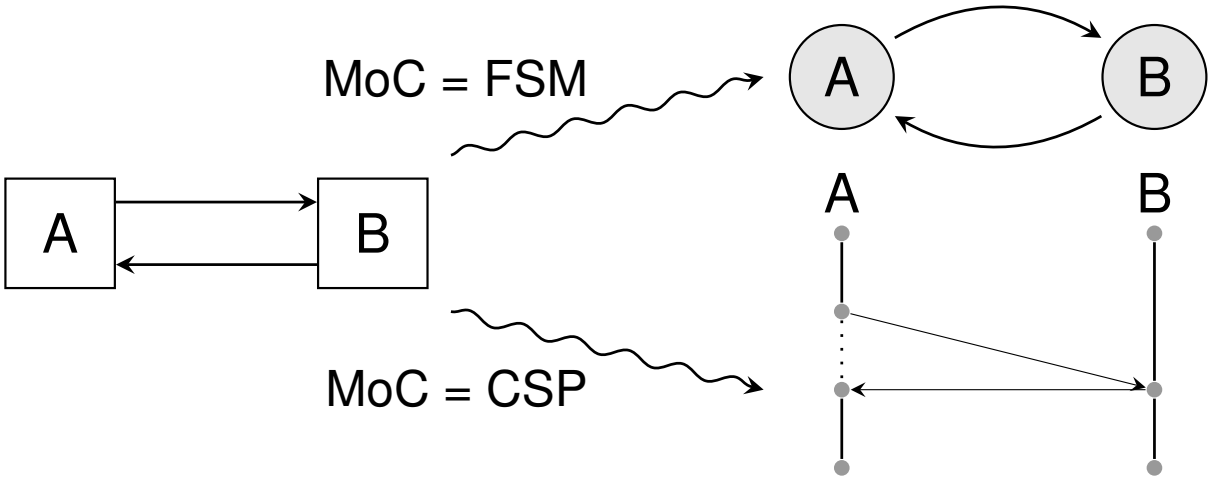


Modèle de calcul



- `react` = tester les transitions qui partent de l'état courant
- `send` = produire un symbole
- `receive` = recevoir un symbole
- `hasData()` = test la présence d'un symbole

Modèle de calcul



- `react` = exécuter le code du processus
- `send` = établir un rendez-vous pour transmettre une donnée
- `receive` = établir un rendez-vous pour recevoir une donnée
- `hasData()` = non implémenté

Sémantique abstraite et modèles de calcul

Caractéristiques

- syntaxe abstraite commune pour tous les modèles
- moteur d'exécution commun à tous les modèles
- sémantique concrète donnée par un modèle de calcul

Avantages

- possibilité d'exécuter des modèles hétérogènes
- possibilité d'ajouter de nouveaux modèles de calcul
- ajout de nouveaux blocs aisé (boîtes noires)

Sémantique abstraite et modèles de calcul

Caractéristiques

- syntaxe abstraite commune pour tous les modèles
- moteur d'exécution commun à tous les modèles
- sémantique concrète donnée par un modèle de calcul

Inconvénients

- codage des modèles de calcul dans la sémantique abstraite
- représentation des modèles avec la syntaxe abstraite
- peu naturel pour certains langages de modélisation

Modélisation par acteurs

Les origines

- années 1970, domaine de l'intelligence artificielle
- Carl Hewitt, Peter Bishop et Richard Steiger (1973)
  A Universal Modular Actor Formalism for Artificial Intelligence
- acteur = élément universel pour le calcul parallèle

Le modèle

- un acteur s'exécute dans son propre flot de contrôle
- il communique avec les autres acteurs par échange de messages

Exemple

Collecte de nourriture par des fourmis
Embouteillages

Modélisation par acteurs

Les acteurs comme constituants de modèles

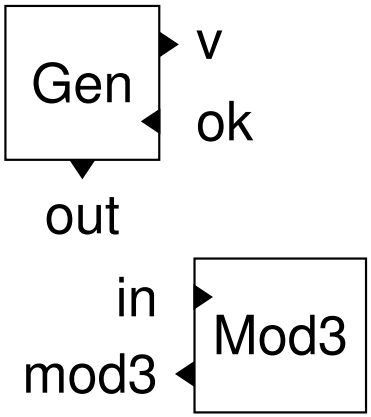
- approche boîte noire
- interface définie par des ports de communication
- sémantique abstraite = protocole des acteurs

Modèle de calcul

- lois de combinaison du comportement des acteurs
- définit :
  - la nature des acteurs (état, processus, opérateur)
  - la nature des relations entre les ports
  - les mécanismes de communication

Influence de la loi de composition

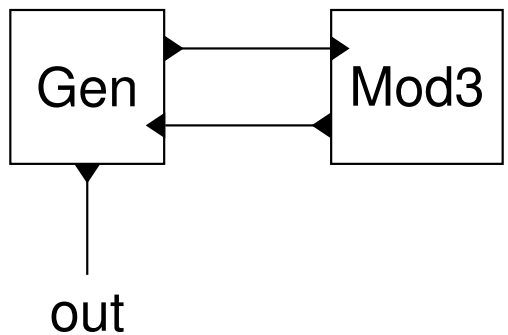
Considérons les composants **Gen** et **Mod3** :



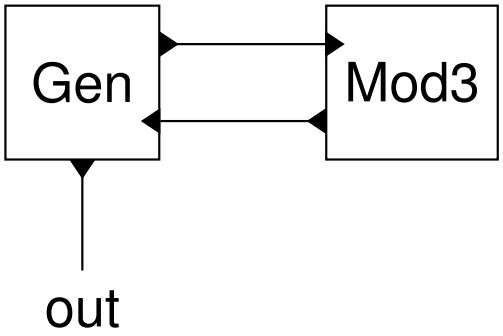
Gen produit la suite des entiers sur **v**, et produit la valeur de **v** sur **out** si la valeur `true` est présente sur son entrée **ok**.

Mod3 produit la valeur `true` sur sa sortie **mod3** si un multiple de 3 est présent sur son entrée **in**.

Le comportement du système construit en connectant ces deux composants comme suit dépend de la manière dont les comportements élémentaires sont composés.

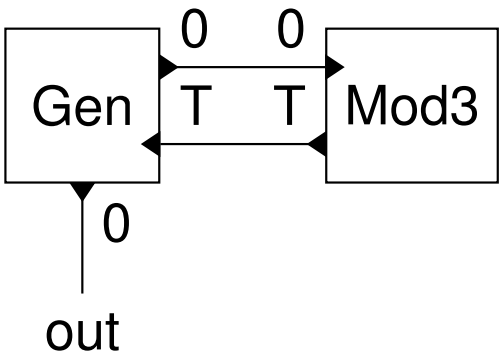


Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

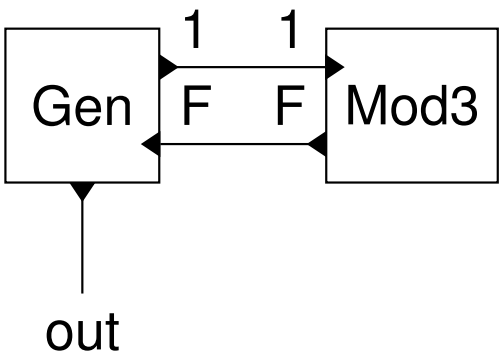
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0								
in	0								
mod3	T								
ok	T								
out	0								

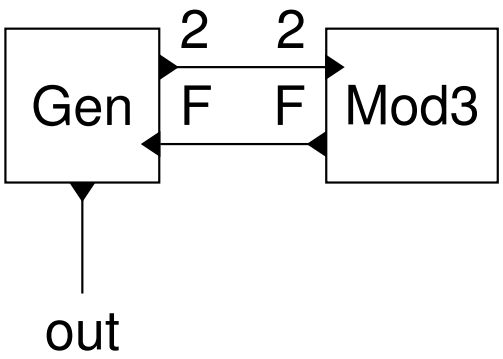
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1							
in	0	1							
mod3	T	F							
ok	T	F							
out	0	$\perp$							

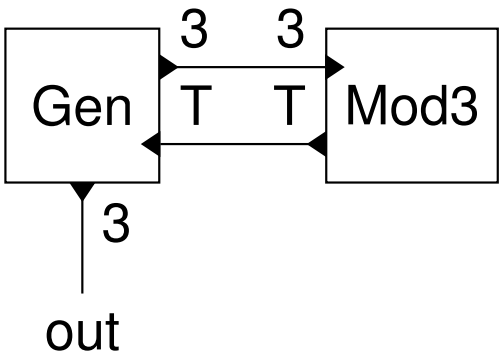
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2						
in	0	1	2						
mod3	T	F	F						
ok	T	F	F						
out	0	$\perp$	$\perp$						

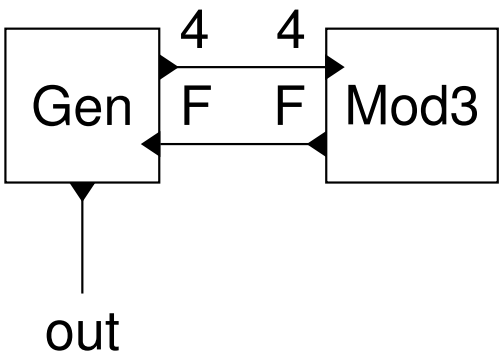
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3					
in	0	1	2	3					
mod3	T	F	F	T					
ok	T	F	F	T					
out	0	$\perp$	$\perp$	3					

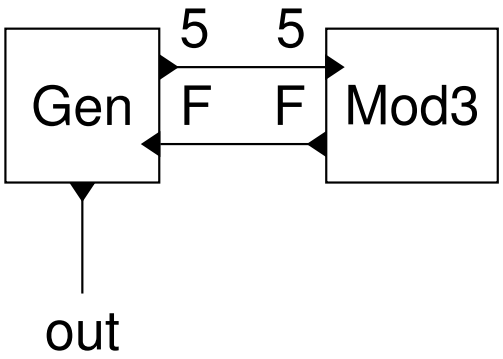
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3	4				
in	0	1	2	3	4				
mod3	T	F	F	T	F				
ok	T	F	F	T	F				
out	0	$\perp$	$\perp$	3	$\perp$				

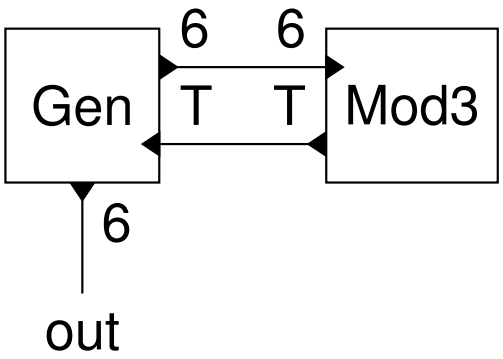
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3	4	5			
in	0	1	2	3	4	5			
mod3	T	F	F	T	F	F			
ok	T	F	F	T	F	F			
out	0	$\perp$	$\perp$	3	$\perp$	$\perp$			

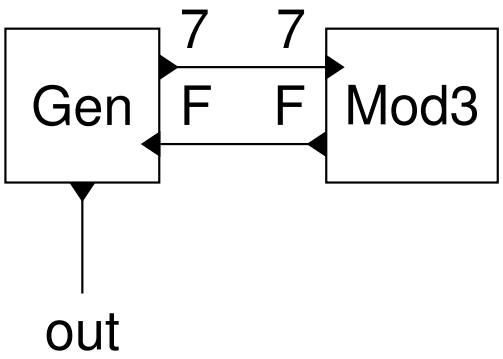
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3	4	5	6		
in	0	1	2	3	4	5	6		
mod3	T	F	F	T	F	F	T		
ok	T	F	F	T	F	F	T		
out	0	$\perp$	$\perp$	3	$\perp$	$\perp$	6		

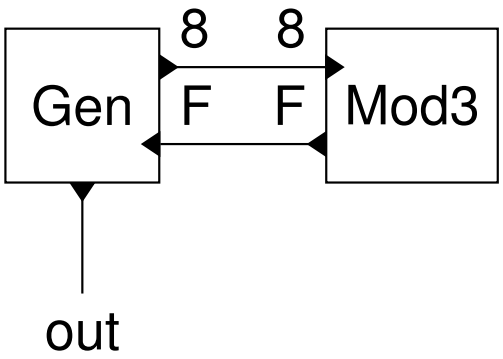
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3	4	5	6	7	
in	0	1	2	3	4	5	6	7	
mod3	T	F	F	T	F	F	T	F	
ok	T	F	F	T	F	F	T	F	
out	0	$\perp$	$\perp$	3	$\perp$	$\perp$	6	$\perp$	

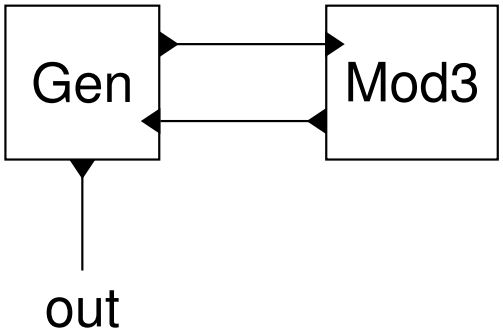
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

v	0	1	2	3	4	5	6	7	8
in	0	1	2	3	4	5	6	7	8
mod3	T	F	F	T	F	F	T	F	F
ok	T	F	F	T	F	F	T	F	F
out	0	$\perp$	$\perp$	3	$\perp$	$\perp$	6	$\perp$	$\perp$

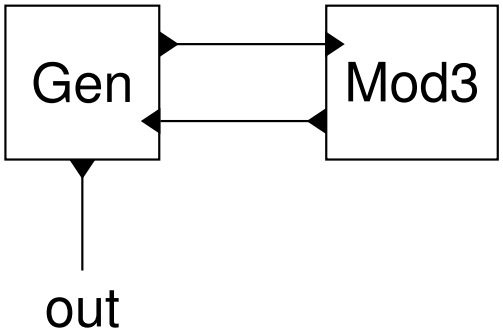
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_n = \text{out}_n$, on observe la suite des multiples de 3 sur **out**.

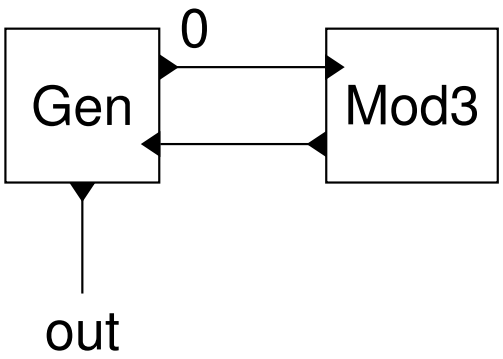
v	0	1	2	3	4	5	6	7	8
in	0	1	2	3	4	5	6	7	8
mod3	T	F	F	T	F	F	T	F	F
ok	T	F	F	T	F	F	T	F	F
out	0	$\perp$	$\perp$	3	$\perp$	$\perp$	6	$\perp$	$\perp$

Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

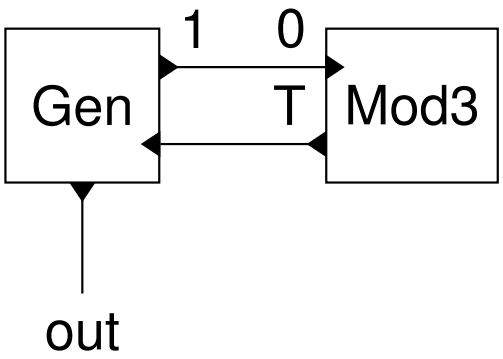
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0								
in	$\perp$								
mod3	$\perp$								
ok	$\perp$								
out	$\perp$								

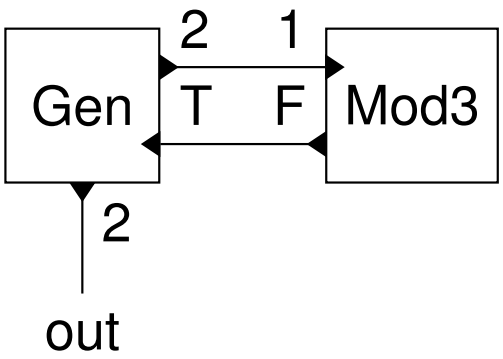
Influence de la loi de composition



- si $\text{out} \longmapsto\!\!\!\!\rightarrow \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1\leq i}$ sur **out**.

v	0	1							
in	$\perp$	0							
mod3	$\perp$	T							
ok	$\perp$	$\perp$							
out	$\perp$	$\perp$							

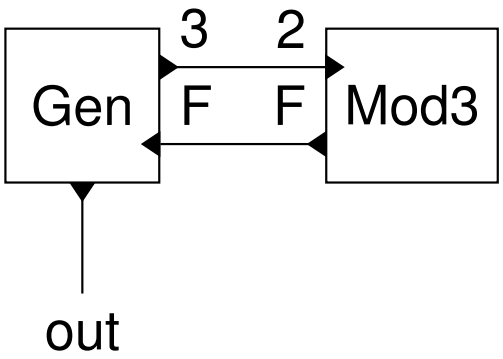
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0	1	2						
in	$\perp$	0	1						
mod3	$\perp$	T	F						
ok	$\perp$	$\perp$	T						
out	$\perp$	$\perp$	2						

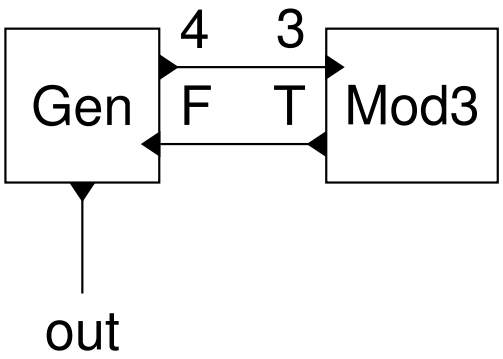
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1\leq i}$ sur **out**.

v	0	1	2	3					
in	$\perp$	0	1	2					
mod3	$\perp$	T	F	F					
ok	$\perp$	$\perp$	T	F					
out	$\perp$	$\perp$	2	$\perp$					

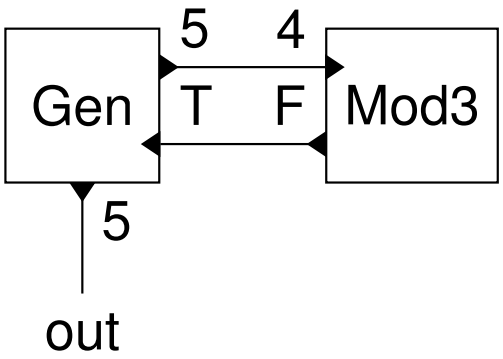
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0	1	2	3	4				
in	$\perp$	0	1	2	3				
mod3	$\perp$	T	F	F	T				
ok	$\perp$	$\perp$	T	F	F				
out	$\perp$	$\perp$	2	$\perp$	$\perp$				

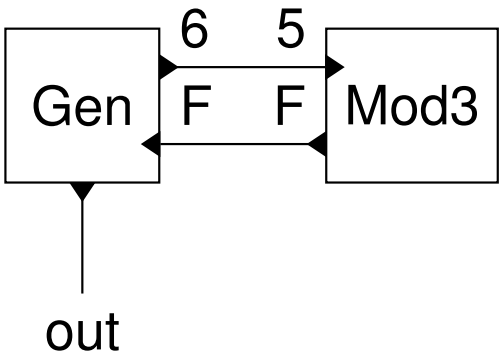
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0	1	2	3	4	5			
in	$\perp$	0	1	2	3	4			
mod3	$\perp$	T	F	F	T	F			
ok	$\perp$	$\perp$	T	F	F	T			
out	$\perp$	$\perp$	2	$\perp$	$\perp$	5			

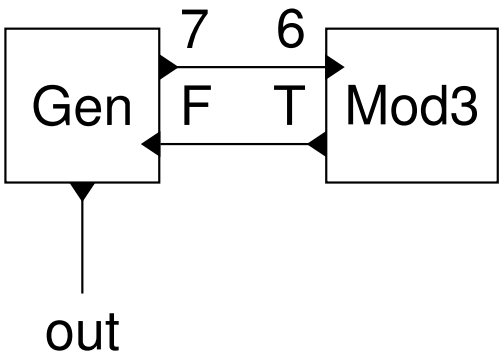
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0	1	2	3	4	5	6		
in	$\perp$	0	1	2	3	4	5		
mod3	$\perp$	T	F	F	T	F	F		
ok	$\perp$	$\perp$	T	F	F	T	F		
out	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$		

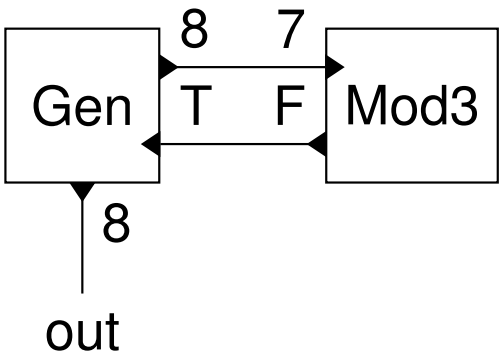
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1\leq i}$ sur **out**.

v	0	1	2	3	4	5	6	7	
in	$\perp$	0	1	2	3	4	5	6	
mod3	$\perp$	T	F	F	T	F	F	T	
ok	$\perp$	$\perp$	T	F	F	T	F	F	
out	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\perp$	

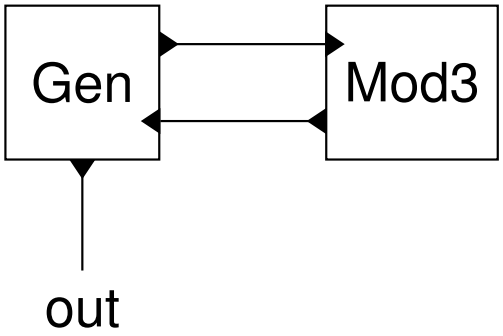
Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1 \leq i}$ sur **out**.

v	0	1	2	3	4	5	6	7	8
in	$\perp$	0	1	2	3	4	5	6	7
mod3	$\perp$	T	F	F	T	F	F	T	F
ok	$\perp$	$\perp$	T	F	F	T	F	F	T
out	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\perp$	8

Influence de la loi de composition



- si $\text{out} \longmapsto \text{in}$ signifie $\text{in}_{n+1} = \text{out}_n$, on observe la suite $(3i-1)_{1\leq i}$ sur **out**.

v	0	1	2	3	4	5	6	7	8
in	$\perp$	0	1	2	3	4	5	6	7
mod3	$\perp$	T	F	F	T	F	F	T	F
ok	$\perp$	$\perp$	T	F	F	T	F	F	T
out	$\perp$	$\perp$	2	$\perp$	$\perp$	5	$\perp$	$\perp$	8

Point clefs pour la sémantique des modèles

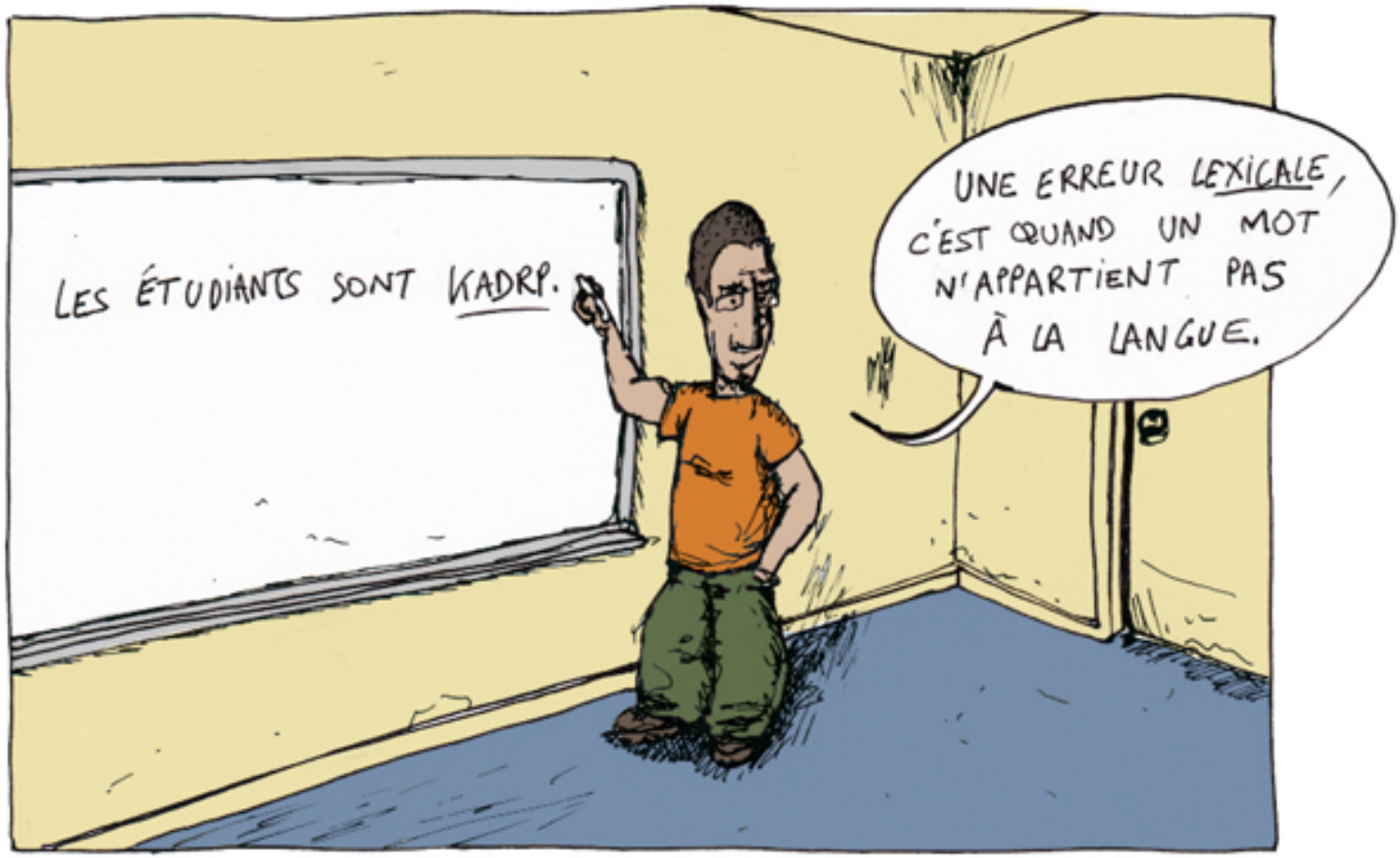
Sémantique

- définition non ambigüe de la syntaxe
- choix d'un domaine sémantique
- définition d'une correspondance
  entre domaines syntaxique et sémantique

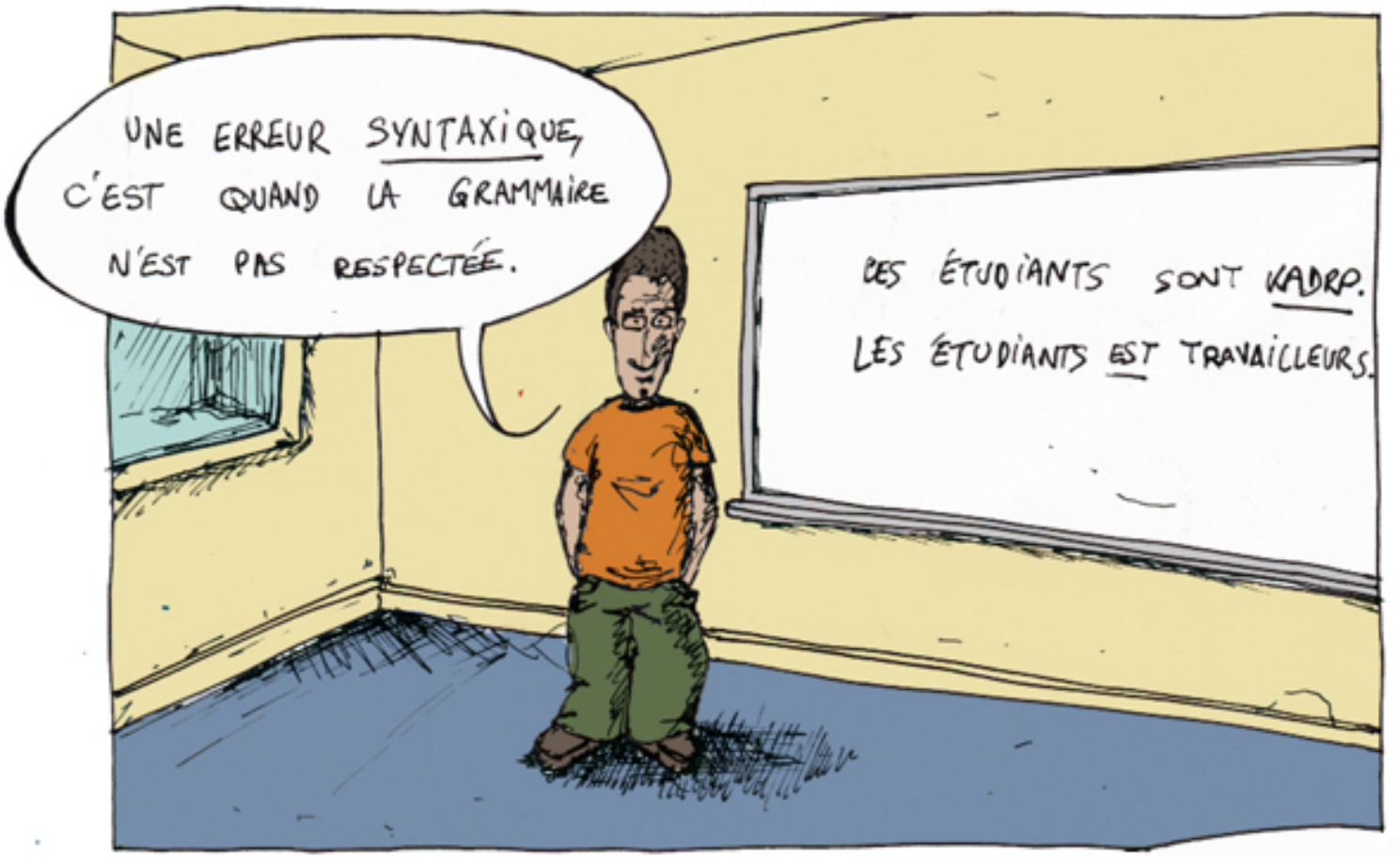
Sémantique abstraite, modèles de calcul

- syntaxe abstraite uniforme
- opérations abstraites pour l'interprétation des modèles
- moteur d'interprétation générique
- sémantique concrète donnée par un MoC
- MoC = loi de composition des comportements

Lexèmes, syntaxe et sémantique



Lexèmes, syntaxe et sémantique



Lexèmes, syntaxe et sémantique