Sémantique élémentaire des langages: Introduction, règles d'inférence et preuves par induction

Didier Buchs

Université de Genève

9 mars 2020

Plan

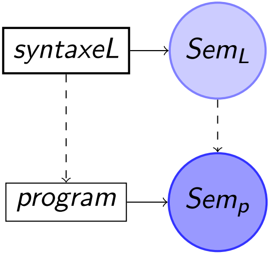
- Pourquoi une sémantique ? Concepts de base.
- Notions de syntaxe abstraite
- Induction mathématique et induction structurelle
- Termes, Types, Termes typés
- Systèmes de transitions
- Différentes sémantiques d'expression arithmétique
  - Sémantique d'évaluation (dénotationnelle)
  - Sémantique computationnelle
  - Sémantique opérationnelle concrète
- Sémantique d'évaluation d'un langage impératif
- Evaluation paresseuse
- Aperçu de sémantique de la concurrence
- Interprétation de programme
- Sémantique de la programmation logique (résolution)

Qu'est ce qu'est une sémantique ?

Sémantique : Relatif au sens (grec sêmantikos : qui signifie)

Objectif : Donner un sens à une description textuelle (fournir un modèle de certains aspects de ce que représente cette description)

- syntaxe = définition des expressions valides
- sémantique = effets de l'évaluation des expressions correctes
  - sur l'état (domaine sémantique)
  - observé par l'ocurrence d'un événement ou de l'expression



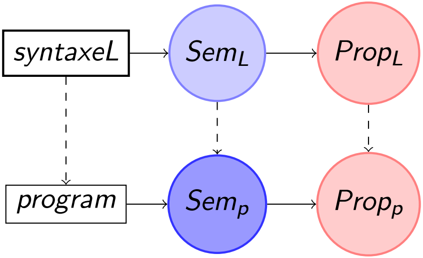
Pourquoi étudier une sémantique ?

- La syntaxe est un domaine bien compris et formalisé, mais qui ne s'intéresse qu'aux propriétés structurelles et grammaticales du langage
- Il n'y a pas de large consensus sur la meilleure méthode pour décrire la sémantique d'un langage, mais il y a un ensemble de méthodes adaptées à chaques types d'objectifs.
- Sémantique statique (typage) et sémantique dynamique (effets de l'exécution)

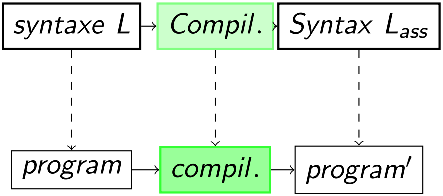
Pourquoi étudier une sémantique ?

- Définir une sémantique pour un langage montre que son utilité est indéniable :
  - pour comprendre comment utiliser le langage
  - pour vérifier qu'un programme répond aux attentes
  - pour compiler correctement les programmes
  - pour transformer (optimiser, paralléliser) les applications

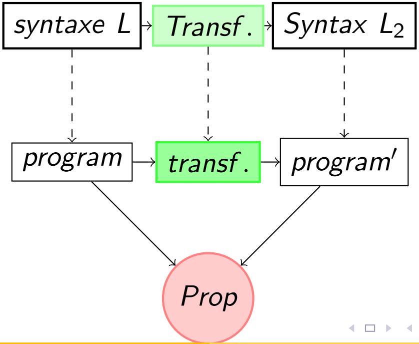
Verifier :



Compiler :



Transformer :



Quels types de sémantique ?

Niveau d'abstraction des sémantiques :

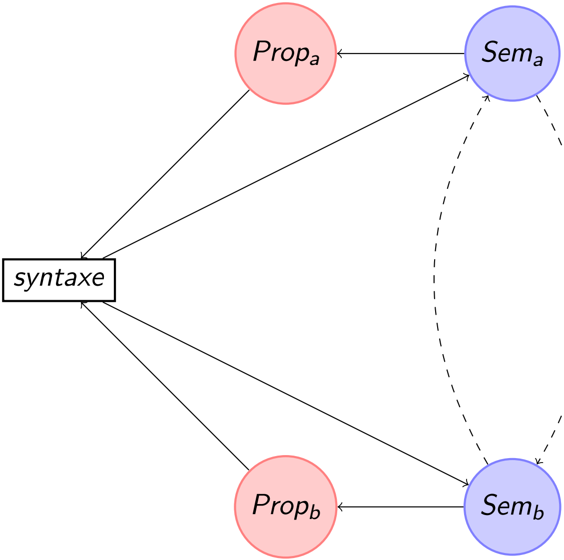
dénotationnelle $>$axiomatique$>$opérationnelle

Pourquoi différents types de sémantiques :

- Référence pour la validité des compilateurs (dénotationnelle)
- Aide à concevoir un langage ($\Rightarrow$ concept simples)(dénotationnelle)
- Référence pour l'apprentissage (dénotationnelle)
- Référence pour les traitement à appliquer sur les programmes (axiomatique)
- Permet la définition d'un compilateur (opérationnelle)

Prend en compte la remarque suivante (Loi de la programmation de Strachey) : *Decide what you want to say before you worry about how you are going to say it*

Relations entre sémantique ?



Qualité de la relations entre sémantique ?

- **complétude** (par rapport à)
  Exemple : Toutes les preuves en logique des clauses de Horn peuvent être prouvées en Prolog
- **validité** (par rapport à)
  Exemple : Toutes les preuves en Prolog sont des preuves valides en logique du 1er ordre

Dans de nombreux cas on se contente de la validité [1].

1. (pour Prolog la sémantique opérationelle est valide par rapport à la sémantique de la logique des clauses de Horn mais pas complète). ! ! Avec la négation ce n'est pas le cas !

Histoire de la sémantique des langages de programmation

- Dénotationnelle : Scott, Strachey, Milne, Stoy 70's pour la sémantique des langages de programmation
- Sémantique du λ-calcul : 1941 Alonzo Church, 1975 langage Scheme, 1990 langage Haskell
- Opérationnelle (SOS) : Plotkin 1960, large utilisation actuellement dans Réseaux de Petri, logiques temporelles, CCS, ...

Syntaxe abstraite

La syntaxe abstraite identifie les composantes significatives des constructions du langage ; elle est liée aux symboles non terminaux de la grammaire.
La syntaxe concrète décrit complètement la représentation écrite (placement des parenthèses, ponctuation, etc)
La même syntaxe abstraite sous-tend ces exemples en Modula-2 et en C :

$$WHILE\ x <> A[i]\ DO\ i := i - 1\ END$$

$$while(x\ != A[i]) i = i - 1;$$

$\Leftrightarrow$ while(**cond**,**prog**)

Syntaxe abstraite vs. syntaxe concrète : opérations arithmétiques

1. $< exp >::=< term > \mid < exp >< lowop >< term >$
2. $< term >::=< num > \mid < term >< highop >< num >$
3. $< lowop >::= +|- \; < highop >::= *|div$
4. $< num >::=< digit > \mid < digit >< num >$
5. $< digit >::= 0|1|2|3|4|5|6|7|8|9$

arbre d'analyse pour : 4 * 2 -1

Syntaxe abstraite vs. syntaxe concrète

1. $< exp >::=< num > \mid < exp >< op >< exp >$
2. $< op >::= + \mid - \mid * \mid div$
3. $< num >::=< digit > \mid < digit >< num >$
4. $< digit >::= 0|1|2|3|4|5|6|7|8|9$

Deux arbres d'analyse pour : 4 * 2 -1

Induction mathématique et induction structurelle

- Preuves par récurrence :
  propriété $P(x)$, pour tout $x \in \mathbb{N}$
- Technique de Preuve :
  - Prouver $P(x)$ est vrai pour $x = 0$ c'est à dire $P(0)$ (cas de base)
  - Supposant que pour tout $k \in \mathbb{N}$, $P(k)$est vrai on prouve que $P(k+1)$ est vrai (cas inductif)
  - Alors P(n) est vrai pour tout $n \in \mathbb{N}$
- Remarque : La validité de la preuve par induction est liée à l'existence d'un ordre bien fondé sur les entiers.

Example

$P(x) = (0 + 1 + 2 + 3... + .. + x - 1 + x = x * (x + 1) div 2)$

Définitions inductives et minimalité

Definition (Nombres Naturels)

- $0 \in \mathbb{N}$
- $x \in \mathbb{N} \Rightarrow x + 1 \in \mathbb{N}$

Mais il existe d'autres ensembles satisfaisant cette propriété : $\mathbb{Q}$, $\mathbb{R}$, ...

Dans sa définition des entiers Peano ajoute une condition d'injectivité à la fonction $+1$, cela élimine également les $\mathbb{Z}^k$

Définitions inductives et minimalité

soit S cet ensemble $S = \{X | 0 \in X \wedge \forall x, x \in X \Rightarrow x+1 \in X\}$

Theorem

$\mathbb{N}$ est le plus petit ensemble satisfaisant ces propriétés,
$\forall X \in S, \mathbb{N} \subseteq X$

Démonstration.

On utilise pour tout $X \in S$: $P_X(n) = (n \in X)$

- $P_X(0)$
- Hyp. $P_X(n)$ vrai, il faut prouver $P_X(n+1)$

$\square$

Définitions inductives d'ensembles

Exemple :

- $0 \in EP$
- $x \in EP \Rightarrow x + 2 \in EP$

Les entiers pairs sont l'ensemble minimum satisfaisant ces propriétés. Il y a d'autres ensembles satisfaisants ces propriétés à l'image des entiers.

$$EP_1 = \{0, 2, 4, 6, ...\}$$

$$EP_2 = \{0, 0.5, 2, 2.5, 4, 4.5, 6, ...\}$$

$$EP_3 = \{0, 0.5, 0.7, 2, 2.5, 2.7, 4, 4.5, 4.7, 6, ...\}$$

...

Définitions inductives d'ensembles

Remarques :

- La propriété $\forall k \in \mathbb{N}, P(k) = 2k \in EP$ est valable pour tous les EP sans être minimal. (en fait la propriété porte sur $\mathbb{N}$ uniquement. Par contre $\mathbb{N}$ est minimal)
  Récurence $2 * 0 = 0 \in EP$ et $2k \in EP$ implique $2 * (k + 1) \in EP$, ici $2 * K + 2$.
- La propriété $\forall n \in EP, P(n) = \forall m \in EP, n + m \in EP$ nécessite l'hypothèse de minimalité. $0.5 + 2.5 = 3 \notin EP_2$

Jugements

Un jugement peut représenter un théorème :

$$\vdash \mathit{theoreme}$$

Les règles de déduction ont la forme classique :

$$\frac{\vdash con_1 \wedge \vdash con_2 \ldots \wedge \vdash con_n}{\vdash conclusion}$$

Les jugements sur les prédicats sont construits sur le même modèle que les jugements en logique.

Définitions d'ensembles :

$$\frac{}{\vdash 0 \in EP} \qquad \forall k, \frac{\vdash k \in EP}{\vdash k + 2 \in EP}$$

$$\forall k, \forall n, \frac{\vdash n \in \mathbb{N}}{\vdash n \in DIV_0} \qquad \forall k, \forall n, \frac{\vdash k \in \mathbb{N} \wedge \vdash n \in DIV_k}{\vdash n \in DIV_{n+k}}$$

Notations pour les définitions inductives

Si le jugement n'a pas de partie gauche, la forme *premisses* $\Rightarrow$ *conclusion* est notée :

$$\frac{\textit{premisses}}{\textit{conclusion}}$$

- les prémisses sont des conjonction de prédicats
- Les prédicats seront construits sur des termes avec ou sans variables
- la conclusion est un prédicat
- Les variables apparaissant dans les prédicats sont implicitement quantifiées universellement.

Exemples

Définitions d'ensembles :

$$\frac{}{0 \in EP} \qquad \frac{k \in EP}{k + 2 \in EP}$$

$$\frac{n \in \mathbb{N}}{n \in DIV_0} \qquad \frac{k \in \mathbb{N}, n \in DIV_k}{n \in DIV_{n+k}}$$

En notation relationelle

$$\frac{n \in \mathbb{N}}{n\ DIV\ 0} \qquad \frac{k \in \mathbb{N}, n\ DIV\ k}{n\ DIV\ n + k}$$

Déductions

- Une déduction est une série d'application de règles de définition inductives
- Les règles se composent 'verticalement' ou une prémisse doit être la conclusion d'une autre règle.
- Les règles des plus haut niveaux sont les règles de bases sans prémisses
- Les variables peuvent être instanciées dans leurs domaines de définitions.

Exemple : prouver que $4 \in EP$

$$\frac{\dfrac{\overline{0 \in EP}}{0+2 \in EP}}{2 + 2 \in EP}$$

Programmation logique en Swift/prolog

Prolog est difficile à combiner avec des langages "classiques"
Solution :

- Interface de type librairie , avec structures de données différentes
- Micro-Kanren, adaptation fonctionelle a la programmation logique
- LogicKit une adaptation a Swift de Prolog

Et Prolog ?

Règles inductives = clauses de Horn, mais aspect opérationnel occulté. Conduit à des problèmes si le prédicat n'est pas réversible. Exemple avec les entiers de prolog : Génération des entiers pairs

Example

```
pair(0).
pair(N) :-pair(M), N is M + 2.
```

test de la parité

Example

```
pairbis(0).
pairbis(N) :- M is N - 2, pairbis(M).
```

Induction structurelle

Les principes d'inductions peuvent s'appliquer à n'importe quelles structures définies inductivements.
Exemple : liste d'entiers naturels : $n \in \mathbb{N}$, $l \in Liste$

Definition

$$\frac{}{[] \in Liste} \qquad \frac{n \in \mathbb{N}, l \in Liste}{n :: j \in Liste}$$

C'est ce que nous définissons comme les termes : $T_{\{[], \_::\_\}}(\mathbb{N})$

Induction structurelle (2)

Exercice : donner une déduction pour la liste : $3 :: 4 :: 1 :: [] \in \mathit{Liste}$

Definition

Une signature est un ensemble d'opérations avec leurs arités :

- OP les noms d'opérations
- $\mu : OP \rightarrow \mathbb{N}$ l'arité des opérateurs

Definition (Termes (non typés)

L'ensemble des termes construit sur un domaine D avec l'ensemble des opérateurs OP (muni d'une arité μ) seront notés $T_{OP}(D)$ Les termes sont définis inductivement par :

- $D \subseteq T_{OP}(D)$
- $\forall op \in OP,\ \mu(op) = n$ l'arité de op et $t_1, \ldots, t_n \in T_{OP}(D)$ $\Rightarrow op(t_1, \ldots, t_n) \in T_{OP}(D)$

Termes

Definition (alternative)

$$\frac{}{D \subseteq T_{OP}(D)} \qquad \frac{t_1 \in T_{OP}(D), ..., t_{\mu(op)} \in T_{OP}(D), op \in OP}{op(t_1, ..., t_{\mu(op)}) \in T_{OP}(D)}$$

Induction structurelle : preuves

Definition

P(x) pour toute liste $\Leftrightarrow$ on peut prouver :

- $P([])$
- $P(l)$ vrai alors $P(n :: l)$ vrai pour tout $n \in \mathbb{N}$

Induction structurelle : exemple

Soit les opérations avec les propriétés usuelles :

- soit max(l) plus grand élément de l
- soit somme(l) somme des éléments de l
- soit long(l) longueur de l

Theorem

$\forall l \in Liste, somme(l) \leq max(l) * long(l)$

Démonstration.

Preuve : $P(x) = somme(x) \leq max(x) * long(x)$
Base : $P([])$ Induction : $P(n :: l)$ sachant que $P(l)$ est vrai $\square$

Définitions des opérations

max(l) somme(l) long(l) $* \leq$

Preuves

Définitions des opérations et relations

0, $succ \equiv s$

- $\overline{0 \in \mathbb{N}}$
- $\dfrac{x \in \mathbb{N}}{s(x) \in \mathbb{N}}$

Définitions des opérations et relations

$*$ $\leq$ definis sur 0, s et $+$

- $\dfrac{x \in \mathbb{N}}{x+0=x}$
- $\dfrac{x,y,k \in \mathbb{N}, x+y=k}{x+s(y)=s(k)}$
- $\dfrac{x \in \mathbb{N}}{x*0=0}$
- $\dfrac{x,y,k,l \in \mathbb{N}, x*y=l, l+n=k}{x*s(y)=k}$
- $\dfrac{}{0 \leq 0}$
- $\dfrac{x \in \mathbb{N}}{0 \leq s(x)}$
- $\dfrac{x,y \in \mathbb{N}, x \leq y}{s(x) \leq s(y)}$
- $\dfrac{x,y \in \mathbb{N}, x=y}{y=x}$ $\dfrac{x \in \mathbb{N}}{x=x}$, $\dfrac{x,y,z \in \mathbb{N}, x=y, y=z}{x=z}$

Définitions des opérations

$max(l) somme(l) long(l)$

- $\dfrac{}{long([])=0}$
- $\dfrac{n\in\mathbb{N}, l\in Liste, long(l)=k}{long(n::l)=k+1}$
- $\dfrac{}{somme([])=0}$
- $\dfrac{n\in\mathbb{N}, l\in Liste, somme(l)=k}{somme(n::l)=k+n}$
- $\dfrac{}{max([])=0}$
- $\dfrac{n\in\mathbb{N}, l\in Liste, max(l)=k, k\leq n}{max(n::l)=n}$
- $\dfrac{n\in\mathbb{N}, l\in Liste, max(l)=k, k\not\leq n}{max(n::l)=k}$

Définitions des principes géneraux des fonctions

f est une fonction,

- $$\frac{f{:}D{*}D{*}...{*}D{\rightarrow}D, t1{=}t1', t2{=}t2', ..., tn{=}tn'}{f(t1,t2,...,tn){=}f(t1',t2',...,tn')}$$

En utilisant la transitivité

- $$\frac{f{:}D{*}D{*}...{*}D{\rightarrow}D, t1{=}t1', t2{=}t2', ..., tn{=}tn', f(t1,t2,...,tn){=}e}{f(t1',t2',...,tn'){=}e}$$

p est un predicat,

- $$\frac{P{:}D{*}D{*}...{*}D, t1{=}t1', t2{=}t2', ..., tn{=}tn', P(t1,t2,...,tn)}{P(t1',t2',...,tn')}$$

Preuves

Démonstration.

Preuve : $P(x) = somme(x) \leq max(x) * long(x)$
Base : $P([])$ Induction : $P(n :: l)$ sachant que $P(l)$ est vrai $\square$

$$P([]) = somme([]) \leq max([]) * long([])$$

$$\dfrac{\dfrac{\dfrac{\overline{0 \leq 0}, \dfrac{\overline{0 \in \mathbb{N}}}{0 * 0 = 0}}{0 \leq 0 * 0}}{somme([]) \leq max([]) * long([])}}{P([])}$$

Démonstration.

Induction : $P(l) \vdash P(n :: l)$ $\square$

Cas 1 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{**1}{somme(l) \leq max(l)*long(l), n \leq max(l) \vdash somme(l)+n \leq max(l)*long(l)+max(l)}}{somme(l) \leq max(l)*long(l), n \leq max(l) \vdash somme(l)+n \leq max(l)*(long(l)+1)}}{somme(l) \leq max(l)*long(l) \vdash somme(l)+n \leq max(n::l)*(long(l)+1)}}{somme(l) \leq max(l)*long(l) \vdash somme(n::l) \leq max(n::l)*long(n::l)}}{P(l) \vdash P(n::l)}$$

**1 est vrai à cause du lemme :

$$\dfrac{a \leq b \vdash c \leq d}{a \leq b \vdash c+a \leq d+b}$$

Cas 2 :

$$\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{\dfrac{**2}{somme(l) \leq max(l)*long(l), max(l) \leq n \vdash somme(l) \leq n*long(l)}}{somme(l) \leq max(l)*long(l), max(l) \leq n \vdash somme(l)+n \leq n*long(l)+n}}{somme(l) \leq max(l)*long(l), max(l) \leq n) \vdash somme(l)+n \leq n*(long(l)+1)}}{somme(l) \leq max(l)*long(l) \vdash somme(l)+n \leq max(n::l)*(long(l)+1)}}{somme(l) \leq max(l)*long(l) \vdash somme(n::l) \leq max(n::l)*long(n::l)}}{P(l) \vdash P(n::l)}$$

**2 est vrai à cause des lemmes :

$$\dfrac{a \leq b \vdash c \leq d}{a \leq b \vdash c*a \leq d*b} \quad \text{et} \quad \dfrac{\vdash c \leq d, \vdash d \leq e}{\vdash c \leq e}$$

Démonstration.

$P(a,b) = \frac{a \leq b \vdash c \leq d}{a \leq b \vdash c+a \leq d+b}$ Cas de base P(0,0) Induction : $P(a,b) \vdash P(a, s(b))$ Induction : $P(a,b) \vdash P(s(a), b)$

$\square$

Cas de base : $\dfrac{P(0{,}0) = \frac{0 \leq 0 \vdash c \leq d}{0 \leq 0 \vdash c+0 \leq d+0}}{P(0{,}0)}$

Cas 1 :

Cas 2 :

Définition de termes

- Les états d'un système peuvent être complexes et construits comme des termes.
- Nous pouvons typer l'ensemble des termes, l'arité sera alors définie sur des noms de types S.
- Un terme libre est construit sur des opérateurs fonctionnels, soit OP l'ensemble de ces opérateurs.
- L'arité (le profil) d'un type est une fonction $\mu : OP \rightarrow S^* \times S$

Exemple :
$S = \{nat, bool\}$
$OP = \{+, -, *, 0, true, false, not\}$
$\mu(+) = (nat\ \ nat, nat)$
$\mu(*) = (nat\ \ nat, nat)$
$\mu(0) = (\epsilon, nat)$
$\mu(not) = (bool, bool)$

Domaines des états, Définition de termes

Definition (Termes)

L'ensemble des termes construit sur un domaine D de type $s \in S$ avec l'ensemble des opérateurs OP (muni d'une arité) seront notés $T_{OP}(D)$ Les termes sont définis inductivement par :

- $D \subseteq T_{OP}(D)$
- $\forall op \in OP$, $\mu(op) = (s_1 s_2 ... s_n, s)$ et $t_1, ..., t_n \in T_{OP}(D)$
  $\forall i, 1 \leq i \leq n, \mu(t_i) = s_i \Rightarrow op(t_1, ..., t_n) \in T_{OP}(D)$

Domaines des états, Définition de termes

Naturellement nous pouvons typer l'ensemble des termes,

Definition (type d'un Terme)

Pour un domaine D de type $s \in S$ et l'ensemble des opérateurs OP la fonction de typage μ des opérations est étendue en $\mu : T_{OP}(D) \rightarrow S$ par :

- $\forall d \in D, \mu(d) = s$
- $\forall op \in OP$, $\mu(op) = (s_1 s_2 ... s_n, s)$ l'arité de op et $t_1, ..., t_n \in T_{OP}(D)$ tel que $\mu(t_1) = s_1, .., \mu(t_n) = s_n$ $\Rightarrow \mu(op(t_1, ..., t_n)) = s$

Exemple : Soit les nombres naturels $\mathbb{N}$ les termes pour les opérateurs $+$ et $*$ sont par exemple :

- $1 + 3 \in T_{\{\_+\_,\_*\_\}}(\mathbb{N})$
- $2 \in T_{\{\_+\_,\_*\_\}}(\mathbb{N})$
- $2 * 3 \in T_{\{\_+\_,\_*\_\}}(\mathbb{N})$
- $(2 * 3) + 4 \in T_{\{\_+,\_*\_\}}(\mathbb{N})$

Exemple : Soit des états représentés par des termes sur les naturels : $State = T_{\{\_+\_,\_*\_\}}(\mathbb{N})$

- $1 + 2 \rightarrow 3$
- $1 + (2 * 2) \rightarrow 1 + 4$
- $1 + 4 \rightarrow 5$
- de manière composée $1 + (2 * 2) \rightarrow 1 + 4 \rightarrow 5$

Syntaxe abstraite EBNF vs. termes

1. $< exp >::=< num > \mid < exp >< op >< exp >$
2. $< op >::= + \mid - \mid * \mid div$
3. $< num >::=< digit > \mid < digit >< num >$
4. $< digit >::= 0|1|2|3|4|5|6|7|8|9$

Comment passer à des termes ?
Chaque domaine syntaxique correspond à des termes différents,

1. exp $= T_{\{\_+\_,\_-\_,\_*\_,\_div\_,\_\}}(num)$,
2. op $= T_{\{+,-,*,div\}}()$,
3. num $= T_{\{'\}}(digit) = T_{\{'\}}(T_{\{0,1,\ldots,9\}}())$,
4. digit $= T_{\{0,1,\ldots,9\}}()$,

Systèmes de transitions

- Les changements d'états d'un système vont être décrit par des systèmes de transitions.
- Il s'agit d'une relation sur les états.
- Les états sont décrit par un ensemble, $State = exp$ par exemple

Definition

Un système de transition est donc une relation sur $State \times State$ notée $\rightarrow$ avec $\rightarrow \subseteq State \times State$

Notation : une transition : $x \in State$ et $y \in State$ et $(x, y) \in State \times State$ sera alors notée $x \rightarrow y$
Exemple : Soit des états représentés par des nombres naturels : $State = \mathbb{N}$

- $1 \rightarrow 3$
- $2 \rightarrow 4 \rightarrow 3$

Systèmes de transitions avec contexte

- Les changements d'états d'un système peuvent dépendre d'un contexte.
- Le contexte est un ensemble contenant la definition des propriétés globales attendues pour la déduction.
- Les états sont décrit par un ensemble, $State = exp$ par exemple
- Un système de transition avec contexte est donc une relation sur : $Context \times State \times State$
- Une transition : $c \in Context$, $x \in State$ et $y \in State$ $(c, x, y) \in Context \times State \times State$ sera alors notée $c \vdash x \rightarrow y$

Exemple : Soit des états représentés par des termes sur les naturels : $State = T_{\{+,*\}}(\mathbb{N})$ et un contexte c :

- $c \vdash 1 + 2 \to 3$
- $c \vdash 1 + (2 * 2) \to 1 + 4$
- $c \vdash 1 + 4 \to 5$

Exemple : Si nous étendons le domaine sur des variables $X = \{A, B\}$: $State = T_{\{+,*\}}(\mathbb{N} \cup X)$ et un contexte $c \in \wp(X \to \mathbb{N})$ attribuant des valeurs aux variables [2] :

- $\{A \to 2\} \vdash 1 + A \to 3$
- $\{A \to 2, B \to 2\} \vdash 1 + (A * B) \to 1 + 4$
- $\{A \to 2, B \to 2\} \vdash 1 + 4 \to 5$

2. également appelé substitution

Systèmes de transitions avec labels

- Les changements d'états d'un système peuvent dépendre d'une action ou d'un événement.
- Les états sont décrit par un ensemble, $State = exp$ par exemple
- Un système de transition avec labels est donc une relation sur : $State \times Label \times State$
- Une transition : $e \in Label$, $x \in State$ et $y \in State$ $(x, e, y) \in State \times Label \times State$ sera alors notée $x \xrightarrow{e} y$

Exemple : Un système producteur consommateur

- Les événements sont : $T_{\{put,get\}}(\mathbb{N})$.
- Les états sont décrit par un ensemble, $State \subseteq \wp(\mathbb{N})$ par exemple.
- Un système de transition pour un producteur consommateur sera :
  - $\{\} \xrightarrow{put(2)} \{2\}$
  - $\{2\} \xrightarrow{put(3)} \{2,3\}$
  - $\{2,3\} \xrightarrow{get(2)} \{3\}$