

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

Didier Buchs

Université de Genève

26 février 2018

- 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)

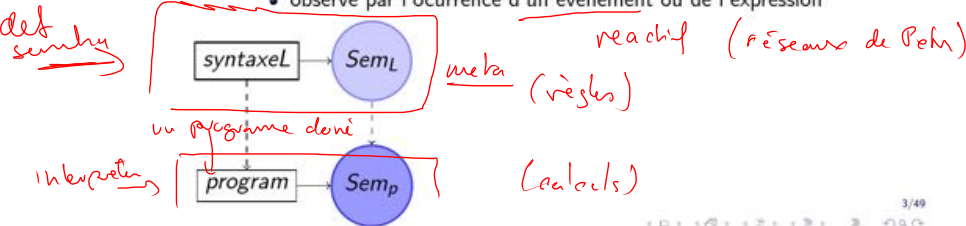
abstrait
concret

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



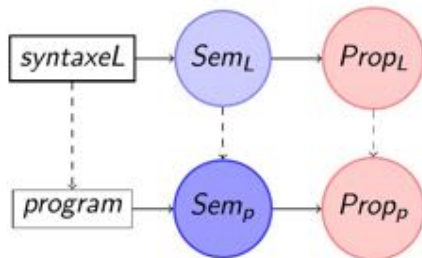
Pourquoi étudier une sémantique ?

- 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 chaque type d'objectifs.
- Sémantique statique (typage) et sémantique dynamique (effets de l'exécution)

Pourquoi étudier une sémantique ?

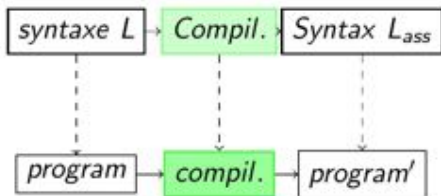
- Définir une sémantique pour un langage montre que son utilité est indéniable :
 - pour comprendre comment utiliser le langage
 - pour vérifier qu'un programme répond aux attentes
 - pour compiler correctement les programmes
 - pour transformer (optimiser, paralléliser) les applications

Vérifier :

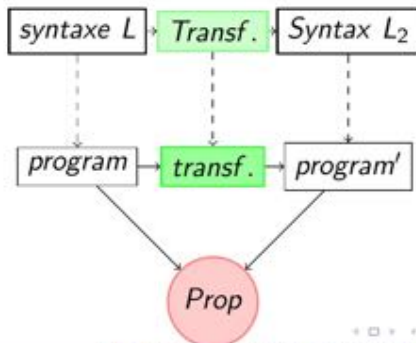


à présent

Compiler :



Transformer :



Quels types de sémantique ?

Niveau d'abstraction des sémantiques :

dénotationnelle > axiomatique > opérationnelle

table de vérité

équivalence

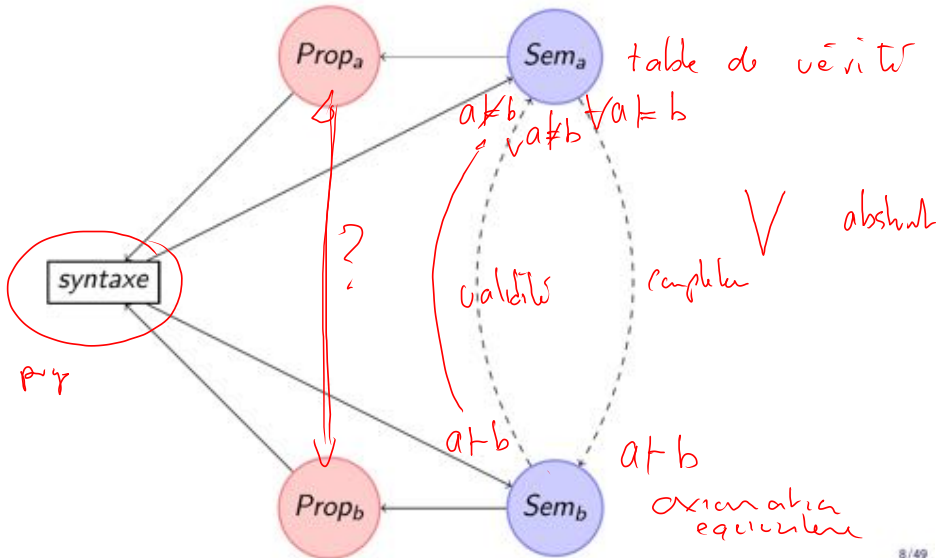
résolution

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 traitements 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 validité¹.

1. (pour Prolog la sémantique opérationnelle 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, ...

Structured Operational Semantics

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 $x \neq A[i]$ *DO* $i := i - 1$ *END*

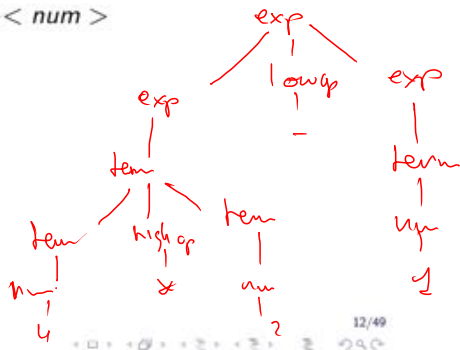
while($x \neq A[i]$) $i = i - 1$;

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

Syntaxe abstraite vs. syntaxe concrète : opérations arithmétiques

- ① $\langle \text{exp} \rangle ::= \langle \text{term} \rangle \mid \langle \text{exp} \rangle \langle \text{lowop} \rangle \langle \text{term} \rangle$
- ② $\langle \text{term} \rangle ::= \langle \text{num} \rangle \mid \langle \text{term} \rangle \langle \text{highop} \rangle \langle \text{num} \rangle$
- ③ $\langle \text{lowop} \rangle ::= + \mid - \quad \langle \text{highop} \rangle ::= * \mid \text{div}$
- ④ $\langle \text{num} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{num} \rangle$
- ⑤ $\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

arbre d'analyse pour : $4 * 2 - 1$



Syntaxe abstraite vs. syntaxe concrète

Syntaxe abstraite

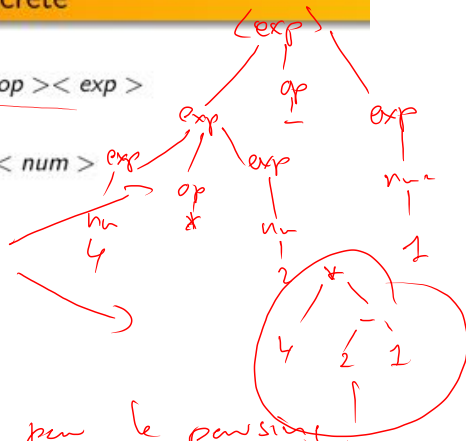
1 $\langle \text{exp} \rangle ::= \langle \text{num} \rangle \mid \langle \text{exp} \rangle \langle \text{op} \rangle \langle \text{exp} \rangle$

2 $\langle \text{op} \rangle ::= + \mid - \mid * \mid \text{div}$

3 $\langle \text{num} \rangle ::= \langle \text{digit} \rangle \mid \langle \text{digit} \rangle \langle \text{num} \rangle$

4 $\langle \text{digit} \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

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



au plus simple

me. pas utilisable pour le parsing.

$$(4 * 2) - 1$$

Induction mathématique et induction structurale

P predicat avec une variable x

- 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

$$P(x) = (0 + 1 + 2 + 3 \dots + \dots + x - 1 + x = x * (x + 1) \text{div} 2)$$

$$\sum_{i=0}^x i = x * (x + 1) \text{div} 2$$

propriétés de l'addition et de la multiplication

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

et la minimalité

Définitions inductives et minimalité

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

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 $P_X(n+1)$



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.

Remarques :

- La propriété $P(k) = 2k \in EP$ est valable pour tous les EP sans être minimal.
Récurrence $2 * 0 = 0 \in EP$ et $2k \in EP$ implique $2 * (k + 1) \in EP$
- La propriété $P(k) = \forall m \in EP, k + m \in EP$ nécessite l'hypothèse de minimalité.

$$E = \{ 0, 0.5, 2, 2.5, \dots \}$$

~~$E \subseteq EP$~~

Un jugement peut représenter un théorème :

$\vdash \textit{theoreme}$

Les règles de déduction ont la forme classique :

$$\frac{\vdash \textit{con}_1 \wedge \vdash \textit{con}_2 \dots \wedge \vdash \textit{con}_n}{\vdash \textit{conclusion}}$$

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

Définitions d'ensembles :

$$\begin{array}{ll} \overline{\vdash 0 \in EP} & \forall k, \frac{\vdash k \in EP}{\vdash k + 2 \in EP} \\ \forall k, \forall n, \frac{\vdash n \in \mathbb{N}}{\vdash n \in DIV_0} & \forall k, \forall n, \frac{\vdash k \in \mathbb{N} \wedge \vdash n \in DIV_k}{\vdash n \in DIV_{n+k}} \end{array}$$

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.

Examples

Définitions d'ensembles :

$$\overline{0 \in EP}$$

$$\frac{n \in \mathbb{N}}{n \in DIV_0}$$

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

$$\frac{k \in \mathbb{N}, n \in \text{DIV}_k}{n \in \text{DIV}_{n+k}}$$

ensubline
and
€ currency -
percent

En notation relationnelle

$$\frac{n \in \mathbb{N}}{n \text{ DIV } 0}$$

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

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 $4 \in EP$

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

↓ de r. ou de d. deduc

Prolog est difficile à combiner avec des langages "classiques"

Solution :

- Interface de type librairie , avec structures de données différentes
- Micro-Kanren, adaptation fonctionnelle 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 inductivement.

Exemple : liste d'entiers naturels : $n \in \mathbb{N}, j \in \text{Liste}$

Definition

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

C'est ce que nous définissons comme les termes :

$$T_{\{[], ::\}}(\mathbb{N})$$

listes d'entiers

opérateur

cas de base

Induction structurelle (2)

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

$$\frac{1 \in \mathbb{N}, \quad \overline{[] \in \text{Liste}}}{}$$

$$\frac{4 \in \mathbb{N}, \quad 1 :: [] \in \text{Liste}}{}$$

$$\frac{3 \in \mathbb{N}, \quad 4 :: (1 :: []) \in \text{Liste}}{}$$

$$3 :: (4 :: (1 :: [])) \in \text{Liste}$$

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

$[] ::$

$$\mu([]) = 0$$

$$\mu(\cdot :: \cdot) = 2$$

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 $\underline{t_1, \dots, t_n} \in \underline{T_{OP}(D)}$
 $\Rightarrow \underline{op(t_1, \dots, t_n)} \in T_{OP}(D)$

(le plus petit ensemble)

Definition (alternative)

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

$T_{\{\emptyset, ::\}}(\mathbb{N}) = \text{liste d'entiers}$

$$\{\{\emptyset, 1\}, 2\} \neq \{\{\emptyset, 2\}, 1\}$$

mais égaux dans les sets

Induction structurelle : preuves

Definition

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

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

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

\uparrow

$\forall n \in \mathbb{N}$

Δ

Induction structurelle : exemple

Soit les opérations avec les propriétés usuelles :

- soit $\max(l)$ plus grand élément de l
- soit $\text{somme}(l)$ somme des éléments de l
- soit $\text{long}(l)$ longueur de l

Theorem

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

Démonstration.

Preuve : $P(x) = \text{somme}(x) \leq \max(x) * \text{long}(x)$

Base : $P([])$ Induction : $P(n :: l)$ sachant que $P(l)$ est vrai



Définitions des opérations

max(l) somme(l) long(l) * ≤

long

$$\text{long}(l) = r$$

?
prédicat d'égalité

$$\text{long}(l) = k, \quad r = s(k)$$

$$\text{long}([]) = 0$$

$$\text{long}(\text{nil}) = r$$

$$\text{long}(l) = k$$

$$\text{long}(\text{nil}) = \underline{\text{succ}(k)}$$

! un calcul

$$\max(l) \quad ?$$

$$\frac{\max([]) = 0}{\text{? arbitraire}}$$

$$\max(l) = k, \quad k \leq n$$

$$\max(n; l) = n$$

$$\max(l) = k, \quad k > n$$

$$\max(n; l) = k$$

$$k > n$$



pas de
non-déterminisme
sur l

$$\text{Somme}(l)$$

Somme ①

$$\frac{\text{Somme}([]) = 0}{\text{Somme}(l) = k}$$

$$\text{Somme}(l) = k$$

$$\text{Somme}(n; l) = k + n$$

↑
?
!

$$\max(l) \neq k_1$$

$$\max(l) = k_2$$

$$k_1 \neq k_2$$

$$0, \quad s x$$

$$\frac{}{0 \in \mathbb{N}}$$

$$\frac{x \in \mathbb{N}}{s(x) \in \mathbb{N}}$$

$$+, \leq$$

$$\frac{\dots}{x + y = k}$$

$$\frac{\begin{array}{l} 0 + x = x \\ y + x = k \end{array}}{s(y) + x = s(k)}$$

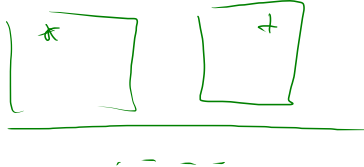
$$\leq_1 \frac{}{0 \leq x}$$

$$\stackrel{\leq}{\textcircled{2}} \frac{x \leq y}{s x \leq s y}$$

$$SSO \leq SO$$

pas de règle applicable !

$$\frac{S(0) \leq 0}{S(S(0)) \leq S(0)} \quad ? \text{ faux}$$



$$x * y = ?$$

$$0 * y = 0$$

$$t * y = k$$

$$s(t) * y = k + y$$

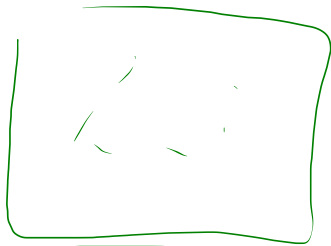
et logique

$$\begin{array}{l} t * y = k, \quad k + y = v \\ \hline s(t) * y = v \end{array}$$

opérateur
à distribuer

Preuves

Cas de base :



Somme ①

$$\text{Somme}([]) = 0, \max([]) * \text{lang}([]) = \text{~~0~~}, 0 \leq \text{~~0~~}$$

Solst 2*

$$\text{Somme}([]) \leq \max([]) * \text{lang}([])$$

$$\underline{a = b, b = c}$$

subs

$$\begin{array}{c} a = c \\ \underline{a = b, f(a) = r} \\ f(b) = r \end{array}$$

$$\begin{array}{c} a = b \quad a \text{ rel } r \\ \hline b \sim r \end{array}$$

transitive

(substitutivité)
 par
 ou par plusieurs variables

(sur les
 relations
 déterministes)

$$\text{Summe}(l) \leq \max(l) * \text{Lang}(l)$$

$$\text{Summe}(u::l) \leq \max(u::l) * \text{Lang}(u::l)$$

Définitions des opérations et relations

0, $\text{succ} \equiv s$

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

Définitions des opérations et relations

* \leq définis sur 0, s et +

$$\bullet \frac{x \in \mathbb{N}}{x+0=x}$$

$$\bullet \frac{x, y, k \in \mathbb{N}, x+y=k}{x+s(y)=s(k)}$$

$$\bullet \frac{x \in \mathbb{N}}{x*0=0}$$

$$\bullet \frac{x, y, k, l \in \mathbb{N}, x*y=l, l+n=k}{\underline{x*s(y)=k}}$$

$$\bullet \overline{0 \leq 0}$$

$$\bullet \frac{x \in \mathbb{N}}{0 \leq s(x)}$$

$$\bullet \frac{x, y \in \mathbb{N}, x \leq y}{s(x) \leq s(y)}$$

$$\bullet \frac{x, y \in \mathbb{N}, x=y}{y=x} \quad \frac{x \in \mathbb{N}}{x=x} \quad \frac{x, y, z \in \mathbb{N}, x=y, y=z}{x=z}$$

symétrie

réflexivité

** transitivité*

Définitions des opérations

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

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

Définitions des principes généraux des fonctions

f est une fonction,

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

✓ substitutivité

En utilisant la transitivité

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

(transitivité
+ substitutivité)

p est un predicat,

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

predicat

Démonstration.

Preuve : $P(x) = \text{somme}(x) \leq \text{max}(x) * \text{long}(x)$

Base : $P([])$ Induction : $P(n :: l)$ sachant que $P(l)$ est vrai □

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

$$\frac{\frac{0 \in \mathbb{N} \quad 0 < 0 \quad 0 + 0 = 0}{0 < 0 + 0}}{\frac{\text{somme}([]) \leq \text{max}([]) * \text{long}([])}{P([])}}$$

Démonstration.

Induction : $P(l) \vdash P(n :: l)$ □

Cas 1 :

$$\begin{array}{l}
 \text{somme}(l) \leq \max(l) * \text{long}(l), n < \max(l) \vdash \text{somme}(l) + n \leq \max(l) * \text{long}(l) + \max(l) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l), n < \max(l) \vdash \text{somme}(l) + n < \max(l) * (\text{long}(l) + 1) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l) \vdash \text{somme}(l) + n \leq \max(n :: l) * (\text{long}(l) + 1) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l) \vdash \text{somme}(n :: l) \leq \max(n :: l) * \text{long}(n :: l)
 \end{array}$$

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

hypothèse du 1^{er} cas $\max(l)$

dist ~~istrib~~

règle de +

avec

succ a de

****1** est vrai à cause du lemme :

$$\begin{array}{l}
 a < b \vdash c < d \\
 a < b \vdash c + a \leq d + b
 \end{array}$$

Cas 2 :

$$\begin{array}{l}
 \text{somme}(l) \leq \max(l) * \text{long}(l), \max(l) \leq n \vdash \text{somme}(l) \leq n * \text{long}(l) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l), \max(l) \leq n \vdash \text{somme}(l) + n \leq n * \text{long}(l) + n \\
 \text{somme}(l) \leq \max(l) * \text{long}(l), \max(l) \leq n \vdash \text{somme}(l) + n \leq n * (\text{long}(l) + 1) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l) \vdash \text{somme}(l) + n \leq \max(n :: l) * (\text{long}(l) + 1) \\
 \text{somme}(l) \leq \max(l) * \text{long}(l) \vdash \text{somme}(n :: l) \leq \max(n :: l) * \text{long}(n :: l)
 \end{array}$$

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

****2** est vrai à cause des lemmes :

$$\begin{array}{l}
 a < b \vdash c < d \\
 a \leq b \vdash c + a \leq d + b
 \end{array}
 \text{ et }
 \begin{array}{l}
 \vdash c \leq d, \vdash d \leq e \\
 \vdash c \leq e
 \end{array}$$

Démonstration.

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



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

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 :

$S = \{nat, bool\}$

$OP = \{+, -, *, 0, true, false, not\}$

$\mu(+) = (nat \ nat, nat)$

$\mu(*) = (nat \ nat, nat)$

$\mu(0) = (\epsilon, nat)$

$\mu(not) = (bool, bool)$

μ sur S donne
co-domaine
 $+$: $_ \rightarrow _$

$\mu(and) = (bool \ bool, bool)$ $\mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$

(S, OP, μ) définissent les opérateurs et leurs arités

$$T_{OP,S} = \{ 0, \text{true}, \text{false}, \text{not}(\text{true}), \\ \cancel{\text{not}(0)}, \text{not}(\text{false}), \text{succ}(0), \dots \\ \dots + (0, \text{succ}(0)), \dots \}$$

$$\frac{C \in OP, N(C) = (\varepsilon, s), s \in S}{C \in (T_{OP,S})_s}$$

lemma
type

$$t_n \in (\text{TOP}, s)_{s_n}$$

$$f \in \text{OP}, \mu(f) = (w, s), w = s_1 \dots s_n, t_1 \in (\text{TOP}, s)_{s_1}$$

$$f(t_1 \dots t_n) \in (\text{TOP}, s)_s$$

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 :

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

valeurs
de base
sur lesquels

sont construits
les termes

$T_{OP,s}$ (ensemble de base)

renvoie

x est un symbole cariable

$T_{OP}(x)$

$T_{OP}(w)$

41/49

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 :

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

$$\begin{aligned} \mu(\leq) &= (\text{nat}, \text{nat}, \text{bool}) \\ \mu(\text{not}(\text{false})) &= \text{bool} \end{aligned}$$

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

$$a + b \Leftarrow + (a, b)$$

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

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

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

Concrètement

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

Syntaxe abstraite EBNF vs. termes

- ① $\langle exp \rangle ::= \langle num \rangle \mid \langle exp \rangle \langle op \rangle \langle exp \rangle$
- ② $\langle op \rangle ::= + \mid - \mid * \mid div$
- ③ $\langle num \rangle ::= \langle digit \rangle \mid \langle digit \rangle \langle num \rangle$
- ④ $\langle digit \rangle ::= 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

EBNF

Comment passer à des termes ?

Chaque domaine syntaxique correspond à des termes différents,

① $exp = T_{\{+, -, *, div\}}(num),$

② ~~$op = T_{\{+, -, *, div\}}(),$~~

③ $num = T_{\{\prime\}}(digit) = T_{\{\prime\}}(T_{\{0,1,\dots,9\}}()), = T_{\{\prime\}}(\{0, \dots, 9\})$

④ ~~$digit = T_{\{0,1,\dots,9\}}(),$~~

$\{0, 1, 2, \dots, 9\}$

opérateur de séquence

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 :

$State = \mathbb{N}$

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

Termes

Systèmes de transitions avec contexte

- Les changements d'états d'un système peuvent dépendre d'un contexte.
- Le contexte est un ensemble contenant la définition des propriétés globales attendues pour la déduction.
- Les états sont décrits 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 $c \vdash x \rightarrow y$

termo

temp

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

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

Exemple : Si nous étendons le domaine sur des variables $X = \{A, B\}$: $State = T_{\{+,*\}}(\mathbb{N} \cup X)$ et un contexte $c \in \wp(X \rightarrow \mathbb{N})$ attribuant des valeurs aux variables² :

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

Systèmes de transitions avec labels

- Les changements d'états d'un système peuvent dépendre d'une action ou d'un événement.
- Les états sont décrits 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 $x \xrightarrow{e} y$

label (^{new} ~~was~~ ^{dis} ~~is~~)

Exemple : Un système producteur consommateur

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