Sémantique de langages de programmation

Master 1 - LC - MPRI

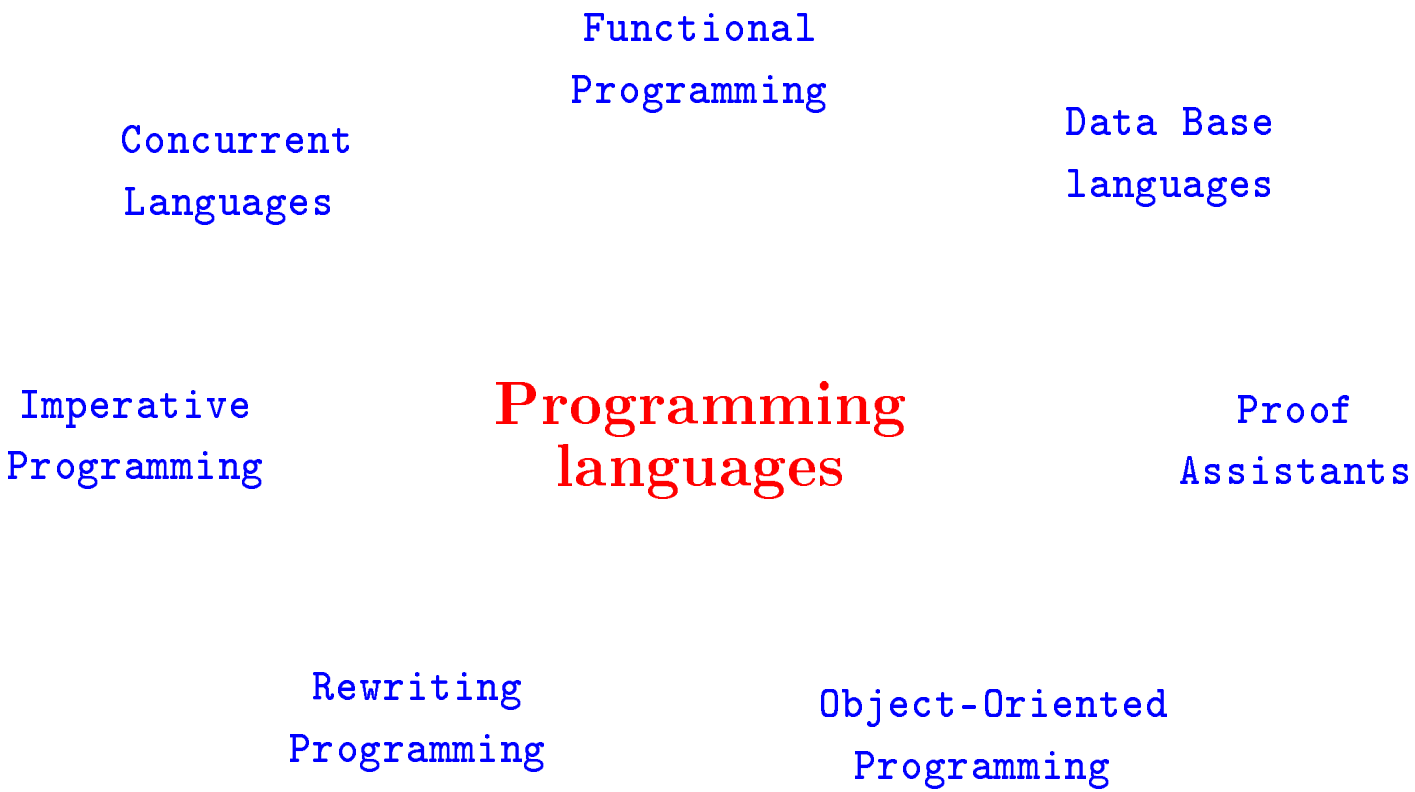
Delia Kesner
PPS, Université Paris VII

Email : kesner@pps.jussieu.fr
URL : www.pps.jussieu.fr/~kesner

Informations générales

- Cours : lundi 10h30-12h30, salle J4
- TD : lundi 12h30-14h30, salle J4
- Partiel Obligatoire : mars
- Note module : (Partiel + Examen)/ 2
- 2eme session : Examen

Motivations



Syntax vs Semantics

Syntax :

$$\forall x \; s(x) \neq 0$$

Semantics :

"Every positive natural number is different from zero"

Semantics of programming languages

The meaning of programming languages.

Different techniques :

- Operational semantics
- Denotational semantics
- Axiomatic semantics

Operational semantics

The meaning of a program is the sequence of states of the machine executing the program.

program = transition system

Example :

```
P1: a=1; b=0;      P2: a=1;          P3: b=0; a=1;
                       b=0;
```

P1 is equivalent to P2, but P1 is not equivalent to P3.

☹ non equivalent programs may have the same result.

This notion can be refined in order to observe just a piece of memory of the machine.

Denotational semantics

program = mathematical function

Argument of this function : the state of the memory before execution

Result of this function : the state of the memory after execution

Example :

```
P1:a=1; b=0;     P2:a=1;      P3:b=0; a=1;    P4:a=1; b=1;
                      b=0;
```

P1, P2 and P3 are equivalent, but they are not equivalent to P4.

Axiomatic semantics

program = transformation of logical properties

This can be written for example in Hoare logic as

$$\{P\}Prog\{Q\}$$

Thus for example,

$$\begin{array}{lll} \{vrai\} & a=1; b=0; & \{a \geq 0 \wedge b \geq 0\} \\ \{vrai\} & b=0; a=1; & \{a \geq 0 \wedge b \geq 0\} \\ \{vrai\} & a=1; b=1; & \{a \geq 0 \wedge b \geq 0\} \end{array}$$

are all equivalent.

Relation between different semantics

Two syntactic equivalent programs are operational equivalent.

Two operational equivalent programs are denotational equivalent.

Two denotational equivalent programs are axiomatic equivalent.

The converse implications are in general false.

Operational Semantics

program = transition system

Rewriting systems are a natural tool to model transitions between objects.

Example I : String rewriting

Rewriting system :

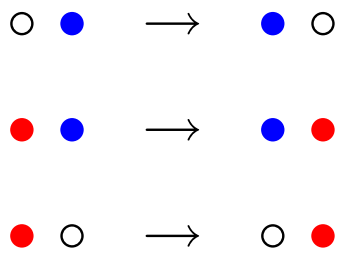
$$\begin{aligned} vert &\longrightarrow orange \\ orange &\longrightarrow rouge \\ rouge &\longrightarrow vert \end{aligned}$$

Reduction sequence :

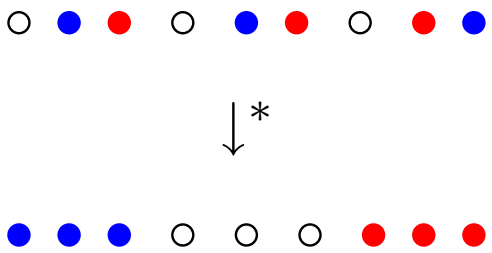
$$vert \longrightarrow orange \longrightarrow rouge \longrightarrow vert \longrightarrow orange \longrightarrow \ldots$$

Example II : String rewriting

Rewriting system :



Reduction Sequence :



Example III : Term rewriting

Rewriting system for Peano arithmetic :

$$\begin{aligned} 0 + y &\rightarrow y \\ s(x) + y &\rightarrow s(x + y) \\ 0 * y &\rightarrow 0 \\ s(x) * y &\rightarrow (x * y) + y \end{aligned}$$

Reduction sequence :

$$
\begin{aligned}
&\text{“}2 * 3\text{”} = \underline{s(s(0)) * s(s(s(0)))} && \rightarrow \\
&\underline{s(0) * s(s(s(0)))} + s(s(s(0))) && \rightarrow \\
&\underline{0 * s(s(s(0)))} + s(s(s(0))) + s(s(s(0))) && \rightarrow \\
&\underline{0 + s(s(s(0)))} + s(s(s(0))) && \rightarrow \\
&\underline{s(s(s(0))) + s(s(s(0)))} && \rightarrow \\
&s(\underline{s(s(0)) + s(s(s(0)))}) && \rightarrow \\
&s(s(\underline{s(0) + s(s(s(0)))})) && \rightarrow \\
&s(s(s(\underline{0 + s(s(s(0)))}))) && \rightarrow \\
&s(s(s(s(s(s(0)))))) = \text{“}6\text{”}
\end{aligned}
$$

Example IV : Equational Programming

Rewriting system :

$$\begin{aligned} nil[a/y] &\rightarrow nil \\ cons(a,x)[a/y] &\rightarrow cons(y, x[a/y]) \\ cons(b,x)[a/y] &\rightarrow cons(b, x[a/y]) \end{aligned}$$

Reduction sequence :

$$\begin{aligned} &cons(a, cons(b, cons(a, cons(b, nil))))[a/c][b/d] \rightarrow^* \\ &cons(c, cons(d, cons(c, cons(d, nil)))) \end{aligned}$$

Example V : Logical reasoning

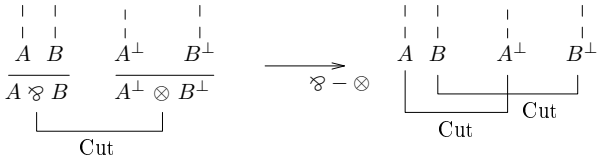
Rewriting system :

$$
\begin{array}{lcl}
p \Rightarrow q & \longrightarrow & \neg p \vee q \\
\neg(p \wedge q) & \longrightarrow & \neg p \vee \neg q \\
\neg(p \vee q) & \longrightarrow & \neg p \wedge \neg q \\
\neg\neg p & \longrightarrow & p
\end{array}
$$

Reduction sequence :

$$
\begin{aligned}
&\neg(\neg(\underline{\neg p \Rightarrow q}) \Rightarrow r) && \longrightarrow \\
&\neg(\neg(\underline{\neg\neg p} \vee q) \Rightarrow r) && \longrightarrow \\
&\neg(\underline{\neg(p \vee q)} \Rightarrow r) && \longrightarrow \\
&\neg(\underline{(\neg p \wedge \neg q) \Rightarrow r}) && \longrightarrow \\
&\neg(\underline{\neg(\neg p \wedge \neg q)} \vee r) && \longrightarrow \\
&\neg((\underline{\neg\neg p} \vee \neg\neg q) \vee r) && \longrightarrow \\
&\neg((p \vee \underline{\neg\neg q}) \vee r) && \longrightarrow \\
&\underline{\neg((p \vee q) \vee r)} && \longrightarrow \\
&\underline{\neg(p \vee q)} \wedge \neg r && \longrightarrow \\
&(\neg p \wedge \neg q) \wedge \neg r
\end{aligned}
$$

Example VI : Graph Rewriting



Example VII : Functional Programming

The λ-calculus [Church]

$$
\begin{array}{lll}
(\beta) & (\lambda x.M)N & \rightarrow \quad M\{x/N\} \\
(\eta) & \lambda x.M\ x & \rightarrow \quad M \\
(SP) & \langle \pi_1(M), \pi_2(M) \rangle & \rightarrow \quad M
\end{array}
$$

Example VIII : Object-Oriented Programming

The ς-calculus [Abadi & Cardelli]

An objet O is a collection of methods $\langle l_i \equiv \varsigma\ self_i.B_i \rangle_{i \in 1...n}$.

Method invocation is given by the rewriting rules :

$$O.l_j \rightarrow B_j\{self_j/O\}$$

Example IX : Mathematical reasoning

$$\exists\alpha.\forall\beta.X \quad \rightarrow \quad \forall\beta.\exists\alpha.X$$

$$T \in \{\alpha : A \mid P\} \quad \rightarrow \quad T \in A \wedge P\{\alpha/T\}$$

Example X : Pattern Matching

$$
\begin{array}{lcl}
case(c_1(L), \lambda x_1.M_1, \ldots, \lambda x_n.M_n) & \rightarrow & M_1\{x_1/L\} \\
\vdots & & \vdots \\
case(c_n(L), \lambda x_1.M_1, \ldots, \lambda x_n.M_n) & \rightarrow & M_n\{x_n/L\}
\end{array}
$$

Example XI : OCAML Programs

```
let rec length l = match l with
    []    -> 0
  | h::t -> length t + 1 ;;

let rec append l1 l2 = match l1 with
    []    -> l2
  | h::t -> h::append t l2;;

let rec map l f = match l with
    []    -> []
  | h::t -> (f h):: map t f ;;
```

Example XII : Concurrent Programming

The π-calculus [Milner & Parrow & Walker]

$$\overline{\alpha}\langle t\rangle.P \mid \alpha\langle x\rangle.Q \quad \rightarrow \quad P \mid Q\{x/t\}$$

Example XIII :XSLT

A language to transform (by rewriting) XML documents.

See `http://www.w3.org/Style/XSL/`

Example XIV : development of plant structures

See `http://algorithmicbotany.org/` for more details.

Formal definition of rewriting

Given a set of objects A, (abstract) rewriting is a relation $\rightarrow \subseteq A \times A$.

String rewriting :	A is a set of words.
First-order rewriting :	A is a set of algebraic terms.
Higher-order rewriting :	A is a set of higher-order terms.
Graph rewriting :	A is a set of graphs.

Main components of the rewriting model

- Objects
- Substitution
- Matching

Typical questions concerning a rewriting model

Consider a rewriting sequence

$$t_1 \rightarrow_{\mathcal{R}} t_2 \rightarrow_{\mathcal{R}} t_3 \rightarrow_{\mathcal{R}} \ldots$$

- Is this computation terminating ? (always ? sometimes ?)
- Is there a result (e.g. canonical form) ?
- Is there uniqueness of results ?

(Part of the) History

(Thue-1914)	String rewrite systems
(Church-1936)	Lambda calculus
(Gorn-1967)	Term rewrite systems
(Klop-1980)	Combinatory reduction systems

Plan du cours

- Introduction
- Notions mathématiques
  - Relations bien fondées : définitions et exemples
  - Principe d'induction bien fondée
  - Composition de relations bien fondées : lexicographique, produit, multi-ensemble, etc
  - Notions de réécriture abstraite
  - Notions de Church-Rosser, confluence, confluence faible, confluence forte, normalisation faible et normalisation forte
  - Quelques théorèmes fondamentaux
- Calculs algébriques
  - Termes et Sigma algèbres

  - Homomorphismes et substitutions
  - Théorème de Birkhoff
  - Réécriture
  - Égalité engendrée par une relation de réécriture
  - Quelques techniques pour montrer la confluence
  - Quelques techniques pour montrer la terminaison
- Calculs fonctionnels
  - Introduction au lambda calcul
  - Lambda termes, substitutions, alpha-conversion
  - Réduction beta et égalité beta
  - Confluence du lambda calcul (réductions parallèles)
  - Lambda calcul typé : type, jugement de type, dérivation de typage
  - Propriétés : Unicité, décidabilité et préservation du typage,

normalisation forte du lambda calcul simplement typé (méthode de réductibilité et preuve arithmétique)

- Sémantique opérationnelle
  - Sémantique opérationnelle structurée (petits pas)
  - Sémantique opérationnelle naturelle (grands pas)
  - Exemple évaluation des nombres binaires
  - Sémantique du lambda calcul : appel par nom et par valeur, à petits pas et grands pas
  - Propriétés : déterminisme, correspondance entre petits pas et grands pas, héritage de la préservation de types et de la normalisation forte
- Machines à environnement pour l'appel par nom
  - Évaluateur simple
  - Machine de Krivine

- Sémantique dénotationnelle
  - L'approche dénotationnelle de la sémantique
  - Sémantique d'un langage fonctionnel simple
  - Modéliser la non terminaison : les ordres partiels complets (CPO)

Bibliographie

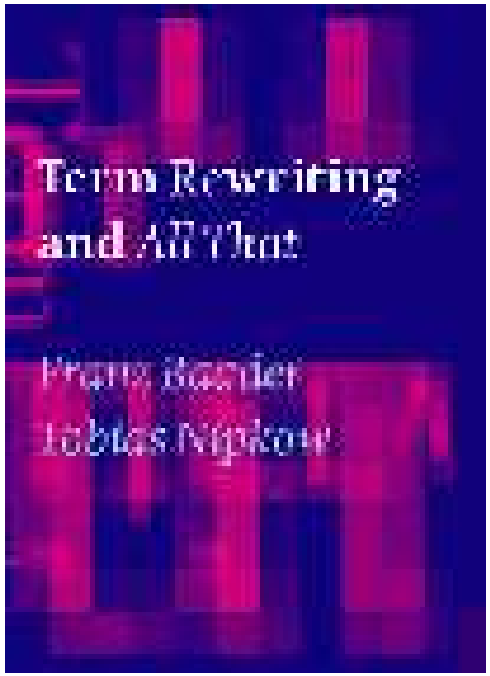
– *Transparents et tableau*

(consulter `www.pps.jussieu.fr/~kesner` régulièrement)

– Pour les calculs algébriques :

Term Rewriting and All That. 1998.

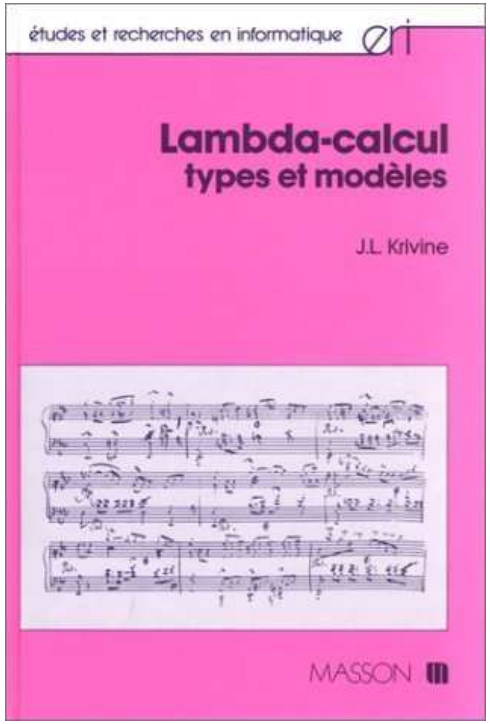
Franz Baader, Tobias Nipkow. Cambridge University Press



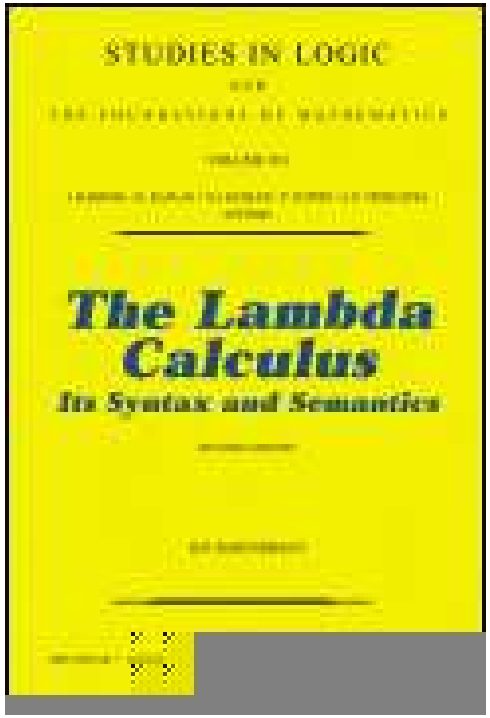
– Pour les calculs fonctionnels :

Lambda-calcul, types et modèles. 1997.

Jean-Louis Krivine. Masson



The lambda calculus : its Syntax and Semantics. 1984. Henk Barendregt. `North-Holland`



Notions mathématiques pour la sémantique

Notions mathématiques de base

Ensembles

Définition : Soient deux ensembles $\mathcal{A}, \mathcal{B}$ inclus dans $\mathcal{U}$.

L'intersection de $\mathcal{A}$ et $\mathcal{B}$ est $\mathcal{A} \cap \mathcal{B} = \{e \in \mathcal{U} \mid e \in \mathcal{A} \text{ et } e \in \mathcal{B}\}$

L'union de $\mathcal{A}$ et $\mathcal{B}$ est $\mathcal{A} \cup \mathcal{B} = \{e \in \mathcal{U} \mid e \in \mathcal{A} \text{ ou } e \in \mathcal{B}\}$

La différence de $\mathcal{A}$ et $\mathcal{B}$ est $\mathcal{A} \setminus \mathcal{B} = \{e \in \mathcal{U} \mid e \in \mathcal{A} \text{ et } e \notin \mathcal{B}\}$

Le complémentaire de $\mathcal{A}$ est $\overline{\mathcal{A}} = \mathcal{U} \setminus \mathcal{A} = \{e \in \mathcal{U} \mid e \notin \mathcal{A}\}$

$\mathcal{P}(\mathcal{A})$ est l'ensemble de toutes les parties de l'ensemble $\mathcal{A}$.

(Lois de *de Morgan*) $\overline{\mathcal{A} \cup \mathcal{B}} = \overline{\mathcal{A}} \cap \overline{\mathcal{B}}$ $\qquad$ $\overline{\mathcal{A} \cap \mathcal{B}} = \overline{\mathcal{A}} \cup \overline{\mathcal{B}}$

Définition : Le produit cartésien de n ensembles $\mathcal{A}_1 \ldots \mathcal{A}_n$ est l'ensemble de n-uplets $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n = \{(a_1, \ldots, a_n) \mid a_i \in \mathcal{A}_i\}$.

Si $\mathcal{A}_i = \mathcal{A}$ pour tout i, on note $\mathcal{A}^n$ le produit $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$.

Relations

Définition : Une relation n-aire sur $\mathcal{A}_1 \ldots \mathcal{A}_n$ est un sous-ensemble de $\mathcal{A}_1 \times \ldots \times \mathcal{A}_n$.

Définition : Soit $R \subseteq \mathcal{A} \times \mathcal{A}$ une relation binaire.

- R est réflexive ssi pour tout $x \in \mathcal{A}$, $(x, x) \in R$.
  R est irréflexive ssi pour tout $x \in \mathcal{A}$, $(x, x) \notin R$.
- R est symétrique si pour tout $x, y \in \mathcal{A}$, $(x, y) \in R$ implique $(y, x) \in R$.
  R est anti-symétrique si pour tout $x, y \in \mathcal{A}$, $(x, y) \in R$ et $(y, x) \in R$ implique $x = y$.
- R est transitive si pour tout $x, y, z \in \mathcal{A}$, $(x, y) \in \mathcal{A}$ et $(y, z) \in R$ implique $(x, z) \in R$.

Notations

- $(x, y) \in R$ peut s'écrire aussi $x \; R \; y$.
- On peut utiliser un symbole à la place de R :

  Ainsi par exemple, si $\leq$ est une relation, alors

  $(x, y) \in \leq$ s'écrit $x \leq y$.
- On écrit $y \geq x$ lorsque $x \leq y$.

Exemples

Example : La relation $\geq$ sur les entiers naturels est réflexive, la relation $>$ sur les entiers naturels est irréflexive.

Example : La relation $=$ sur les ensembles est symétrique, la relation $\geq$ sur les entiers naturels est anti-symétrique.

Example : La relation $\supseteq$ sur les ensembles est transitive.

Équivalence et Congruence

Définition :

- R est une équivalence si elle est réflexive, symétrique et transitive.

  Exercice : Montrer que $\sim = \{(x, y) \mid 3 \text{ est diviseur de } x - y\}$ est une équivalence.
- R est une congruence p.r. à f si R est une équivalence compatible avec f, c'est à dire, si $a_1\ R\ b_1 \ldots a_n\ R\ b_n$ implique $f(a_1, \ldots, a_n)\ R\ f(b_1 \ldots b_n)$.

  Exercice : Montrer que $\sim = \{(x, y) \mid 3 \text{ est diviseur de } x - y\}$ est une congruence par rapport à $+$ et à $*$.

Classes d'équivalence

La classe d'équivalence de $a \in \mathcal{A}$ par rapport à une équivalence R est l'ensemble $[a]_R = \{b \in \mathcal{A} \mid aRb\}$.

Composition de relations

Définition : Si $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{B}$ et $\mathcal{S} \subseteq \mathcal{B} \times \mathcal{C}$, alors la composition de $\mathcal{S}$ avec $\mathcal{R}$ est une relation dans $\mathcal{A} \times \mathcal{C}$ t.q.
$\mathcal{S} \circ \mathcal{R} = \{(x, y) \in \mathcal{A} \times \mathcal{C} \mid \exists z \in \mathcal{B}\ (x, z) \in \mathcal{R}$ et $(z, y) \in \mathcal{S}\}$.

Définition : Soit $\mathcal{R} \subseteq \mathcal{A} \times \mathcal{A}$. On note $\mathcal{R}^n$ la n-composition de $\mathcal{R}$ avec elle même par induction comme suit :

$$\begin{aligned} \mathcal{R}^0 &= \{(a, a) \mid a \in \mathcal{A}\} \\ \mathcal{R}^{n+1} &= \mathcal{R}^n \circ \mathcal{R} = \mathcal{R} \circ \mathcal{R}^n = \underbrace{\mathcal{R} \circ \ldots \circ \mathcal{R}}_{n+1 \text{ fois}} \end{aligned}$$

Example : Soit $A = \{Paris, Lyon, Toulouse\}$ et

$R = \{(Paris, Lyon), (Paris, Toulouse),$
$(Lyon, Paris), (Toulouse, Paris)\},$

$R^2 = \{(Paris, Paris), (Lyon, Lyon), (Toulouse, Toulouse),$
$(Lyon, Toulouse), (Toulouse, Lyon)\},$

Calculer R^3.

Les clôtures

Définition : La clôture transitive d'une relation $\mathcal{R}$ est donnée par

$$\mathcal{R}^+ = \bigcup_{n=1}^{\infty} \mathcal{R}^n$$

La clôture réflexive et transitive d'une relation $\mathcal{R}$ est donnée par

$$\mathcal{R}^* = \bigcup_{n=0}^{\infty} \mathcal{R}^n = \mathcal{R}^+ \cup \mathcal{R}^0$$

Example : Dans l'exemple d'avant, $R^* = A \times A$.

Fonctions

Définition : Une fonction f entre deux ensembles $\mathcal{A}$ et $\mathcal{B}$, notée $f : \mathcal{A} \to \mathcal{B}$, est une relation sur $\mathcal{A} \times \mathcal{B}$ t.q. pour tout x, y, z si $(x, y) \in f$ et $(x, z) \in f$, alors $y = z$.

Notation : On écrit $f(x)$ pour dénoter l'unique élément y t.q. $(x, y) \in f$ et $f(\mathcal{C}) = \{y \in \mathcal{B} \mid \exists x \in \mathcal{C}, f(x) = y\}$.

On note $id_{\mathcal{A}}$ la fonction identité sur $\mathcal{A}$ donnée par $id_{\mathcal{A}}(x) = x$.

Définition : Soit $f : \mathcal{A} \to \mathcal{B}$ une fonction.

pas toujours une fonction

- L'inverse de f est $f^{-1} = \{(y, x) \in \mathcal{B} \times \mathcal{A} \mid (x, y) \in f\}$

Composition de fonctions

Définition :

- La composition de $f : \mathcal{B} \to \mathcal{C}$ avec $g : \mathcal{A} \to \mathcal{B}$ est la fonction $f \circ g : \mathcal{A} \to \mathcal{C}$, où $f \circ g(x) = f(g(x))$.
  Example : $f(x) = x^2$, $g(x) = x + 4$, $f \circ g(x) = (x + 4)^2$, $g \circ f(x) = x^2 + 4$.
- La n-composition de f avec elle-même , notée f^n, est défini par récurrence sur n :
  - Si $n = 0$, alors $f^0 = id$
  - Si $n > 0$, alors $f^n = f \circ f^{n-1}$

  Example : $f(x) = x + 2$, $f^0(x) = x$, $f^1(x) = x + 2$, $f^2(x) = x + 4$, $f^3(x) = x + 6$, ..., $f^n(x) = x + 2.n$.

Par induction, voir

la Section suivante

Exercice : Soit $n > 0$. Montrer que $f^n = f^{n-1} \circ f$.

Propriétés des fonctions

Définition : Une fonction $f : \mathcal{A} \to \mathcal{B}$ est injective ssi pour tout $x, y \in \mathcal{A}$, $f(x) = f(y)$ implique $x = y$.

Example : $f(x) = x + 2$ sur les entiers est injective.
$f(x) = x \setminus \{3\}$ sur les ensembles d'entiers n'est pas injective. Ainsi $f(\{2, 3, 4\}) = f(\{2, 4\})$ mais $\{2, 3, 4\} \neq \{2, 4\}$.

Définition : Une fonction $f : \mathcal{A} \to \mathcal{B}$ est surjective ssi pour tout $y \in \mathcal{B}$ il existe $x \in \mathcal{A}$ tel que $f(x) = y$.

Example : $f(x) = x$ div 2 sur les entiers naturels est surjective.
$f(x) = x + 2$ sur les entiers naturels n'est pas surjective.

Définition : Une fonction est bijective ssi elle est injective et surjective.

Example : Soit $\mathcal{A}$ l'ensemble de mots de longueur 3 contenant uniquement 0 et 1. Soit $\mathcal{B} = \{0 \dots 7\}$. Soit $f(\text{"}b_2b_1b_0\text{"}) = b_2.2^2 + b_1.2^1 + b_0.2_0$. Cette fonction est injective et surjective, donc bijective.

Fonction caractéristique

Définition : Soit $\mathcal{A}$ un ensemble inclus dans un univers $\mathcal{U}$. La fonction caractéristique de $\mathcal{A}$ dans $\mathcal{U}$ est la fonction $\chi : \mathcal{U} \rightarrow \{0, 1\}$ telle que

$$\forall a \in \mathcal{U}.\chi(a) = 1 \text{ ssi } a \in A$$

Préordres, ordres

Définition :

– Un préordre est une relation réflexive et transitive.

Example :

$\mathcal{R} = \{(2,2),(3,3),(4,4),(3,2),(2,3),(2,4),(3,4)\}$.

– Un ordre ou ordre partiel est une relation réflexive, anti-symétrique et transitive.

Notation : $\geq$

Example : $\mathcal{R}$ n'est pas un ordre car $(3,2),(2,3)$ mais $2 \neq 3$.
$\mathcal{S} = \{(2,2),(3,3),(4,4),(2,3),(2,4),(3,4)\}$ est un ordre.

Définition : Un ordre strict est une relation irréflexive et transitive.

Notation : $>$

Example : $>$ sur les entiers, $\supset$ sur les ensembles.

Définition : Un ordre strict est bien fondé ssi il n'existe aucune chaîne infinie (i.e., de la forme $a_0 > a_1 > a_2 > \ldots$).

Example : $>$ sur les entiers naturels est bien fondé. $>$ sur tous les entiers n'est pas bien fondé. $\supset$ sur les ensembles est bien fondé.

Majorants/minorants et bornes supérieures/inférieures

Soit $\mathcal{E}$ un ensemble muni d'un ordre $\leq$. Soit $\mathcal{A} \subseteq \mathcal{E}$.

Définition :

Un majorant de $\mathcal{A}$ est un $x \in \mathcal{E}$ t.q. pour tout $y \in \mathcal{A}$, $y \leq x$.

Un minorant de $\mathcal{A}$ est un $x \in \mathcal{E}$ t.q. pour tout $y \in \mathcal{A}$, $x \leq y$.

La borne supérieure de $\mathcal{A}$, notée $sup(\mathcal{A})$, est le plus petit des majorants de $\mathcal{A}$ (si z est un majorant de $\mathcal{A}$ alors $sup(\mathcal{A}) \leq z$).

La borne inférieure de $\mathcal{A}$, notée $inf(\mathcal{A})$, est le plus grand des minorants de $\mathcal{A}$ (si z est un minorant de $\mathcal{A}$ alors $z \leq inf(\mathcal{A})$).

Example : Soit $\mathcal{A} = \{1, \ldots, 10\}$. Tous les entiers dans $\{10, \ldots\}$ sont des majorants de $\mathcal{A}$ et 10 est la borne supérieure.

Example : Si $a \leq c$, $a \leq d$, $b \leq c$, $b \leq d$, alors a et b sont des minorants, mais comme ils sont incomparables il n'y a pas de borne inférieure.

Example : Si E_i sont des ensembles dans $\mathcal{P}(\mathcal{E})$, alors le sup est $\bigcup_i E_i$ et le inf $\bigcap_i E_i$.

Fonctions monotones et points fixes

Définition : Soit $f : \mathcal{A} \to \mathcal{B}$ une fonction et soient $\leq_{\mathcal{A}}, \leq_{\mathcal{B}}$ deux ordres sur $\mathcal{A}$ et $\mathcal{B}$ respectivement.

La fonction f est monotone ssi $x \leq_{\mathcal{A}} y$ implique $f(x) \leq_{\mathcal{B}} f(y)$.

Example : $f(x) = x + 3$.

Définition : Soit $f : \mathcal{A} \to \mathcal{A}$ une fonction.

Un point fixe de f est un élément $x \in \mathcal{A}$ t.q. $f(x) = x$.

Example : Soit $f(x) = x^2$. Alors $x = 1$ est un point fixe.

Le plus petit point fixe de f est $inf(\{x \in \mathcal{A} \mid f(x) = x\})$.

Le plus grand point fixe de f est $sup(\{x \in \mathcal{A} \mid f(x) = x\})$.

Définitions Inductives

Définitions inductives en informatique

- Syntaxe concrete
- Syntaxe abstraite
- Règles de typage
- Règles d'évaluation

Le principe

Une définition inductive est caracterisée par :

- Une ou plusieures assertions
- Un ensemble de règles d'inférence pour dériver ces assertions

Example :

- Assertion : "`X est naturel`" ou "`X nat`"
- Règles d'inférence :

  R1 : `0 est naturel`

  R2 : Si `n est naturel`, alors `succ(n) est naturel`.

Notation

Les règles d'inférence sont notées

$$\frac{\texttt{Hypothèse}_1 \ldots \texttt{Hypothèse}_n}{\texttt{Conclusion}} \text{ (Nom de la règle)}$$

- `Conclusion` est une assertion
- $\texttt{Hypothèse}_1 \ldots \texttt{Hypothèse}_n$ sont des assertions
- En général $n \geq 0$. Si $n = 0$ la règle est un axiome

Exemple (règle unaire)

Les entiers naturels

$$\frac{}{0 \texttt{ est naturel}}\,(Nat0) \qquad \frac{n \texttt{ est naturel}}{succ(n) \texttt{ est naturel}}\,(Nat+)$$

Exemple (règle binaire)

Les arbres binaires

$$\frac{}{vide \texttt{ est un arbre binaire}}\ \text{(Abin-nil)}$$

$$\frac{A_1 \texttt{ est un arbre binaire} \qquad A_2 \texttt{ est un arbre binaire}}{node(A_1, A_2) \texttt{ est un arbre binaire}}\ \text{(Abin-ind)}$$

Exemple

Les mots sur un alphabet A

$$\frac{}{\epsilon \ \texttt{mot}} \qquad \frac{a \in A \quad n \ \texttt{mot}}{a.n \ \texttt{mot}}$$

Exemple (plusieurs axiomes, règles unaires et binaires)

Les expressions de la logique propositionnelle sur l'alphabet A

$$\frac{p \in A}{p \texttt{ expr}}$$

$$\frac{A_1 \texttt{ expr} \qquad A_2 \texttt{ expr}}{A_1 \vee A_2 \texttt{ expr}} \qquad \frac{A_1 \texttt{ expr} \qquad A_2 \texttt{ expr}}{A_1 \wedge A_2 \texttt{ expr}}$$

$$\frac{A_1 \texttt{ expr} \qquad A_2 \texttt{ expr}}{A_1 \to A_2 \texttt{ expr}} \qquad \frac{A \texttt{ expr}}{\neg A \texttt{ expr}}$$

Exemple (plusieures assertions)

Les forêts de type T

$$\frac{}{avide \in \texttt{arbre T}} \qquad \frac{}{fvide \in \texttt{foret T}}$$

$$\frac{t \in \texttt{T} \quad f \in \texttt{foret T}}{node(t, f) \in \texttt{arbre T}} \qquad \frac{A \in \texttt{arbre T} \quad f \in \texttt{foret T}}{add(A, f) \in \texttt{foret T}}$$

Dérivation d'une assertion

Une assertion A est dérivable ssi

– A est un axiome

$$\frac{}{A}$$

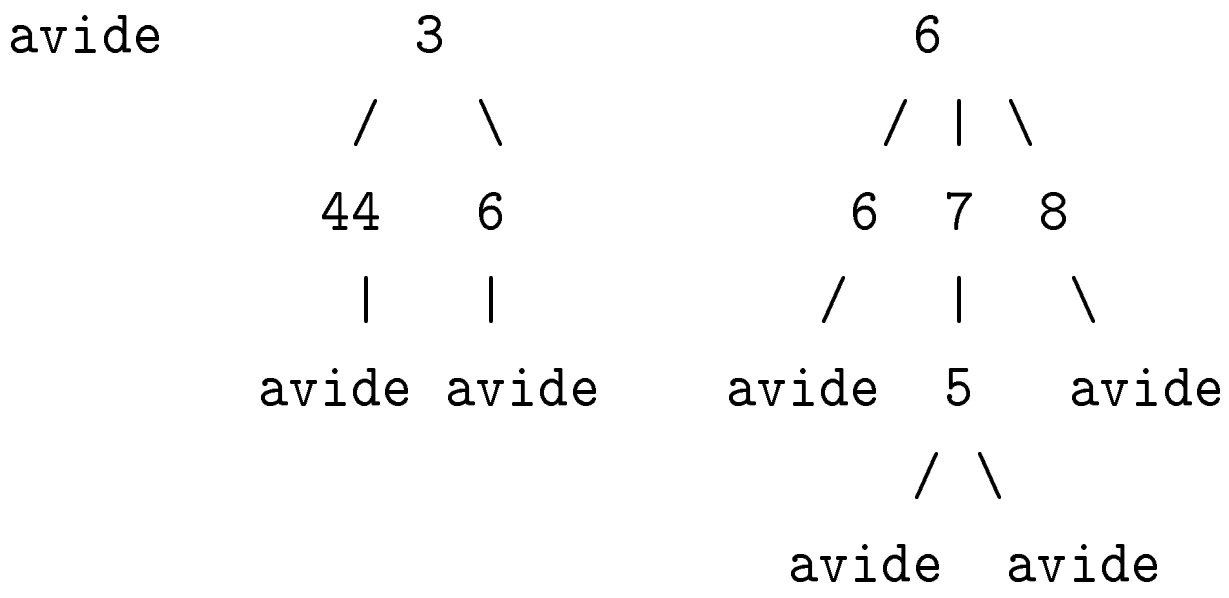
– ou il y a une règle de la forme

$$\frac{A_1 \qquad A_n}{A}$$

telle que $A_1, \ldots, A_n$ sont dérivables

Exercice :

1. Montrer que $succ(succ(succ(0)))$ `nat` est dérivable.
2. Donner le terme qui dénote la forêt suivante et montrer comment la construire avec les règles précédentes :



Ensemble inductif

Un ensemble inductif est le plus petit ensemble engendré par un système de règles d'inférence.