

Cours 6

Logique et Informatique

Programmation (en) logique

Logique – Licence Informatique



Langage logique = Langage de programmation ?

... *pas tout à fait*

- **programmation (en) logique**

- ▶ PROLOG, etc.

exécuter un programme c'est construire une preuve et en extraire de l'information

programme	≡	ensemble de formules logiques
exécution d'un programme	≡	recherche d'une preuve

- pour obtenir un *langage de programmation (en) logique*, on se restreint à certaines formules logiques : les **clauses de Horn**

Clauses

Atome	$p(t_1, \dots, t_n)$ p : predicat t_i : termes	$pair(succ(x))$
Clause	$\forall \vec{X}(\neg A_1 \vee \dots \vee \neg A_n \vee B_1 \vee \dots \vee B_k)$ $\forall \vec{X}((A_1 \wedge \dots \wedge A_n) \Rightarrow (B_1 \vee \dots \vee B_k))$ A_i, B_j : atomes	
Clause de Horn	$\forall \vec{X}(\neg A_1 \vee \dots \vee \neg A_n \vee B)$ $(k \leq 1)$	$(k = 1)$ $(k = 1)$
		Clause définie
	$\forall \vec{X}(\neg A_1 \vee \dots \vee \neg A_n)$ $\neg \exists \vec{X}(A_1 \wedge \dots \wedge A_n)$	$(k = 0)$ $(k = 0)$
		Clause négative

Ecrire un programme avec des « clauses définies »

- exemple : addition de deux entiers :

$$\forall x \text{ add}(0, x, x)$$

$$\forall x \forall y \forall z \text{ add}(x, y, z) \Rightarrow \text{add}(\text{succ}(x), y, \text{succ}(z))$$

- comment « utiliser » un programme (en) logique ?
 - ▶ en soumettant une requête de la forme $\exists \vec{x} (B_1 \wedge \dots \wedge B_k)$
- exemple : additionner 1 avec 2
 - ▶ existe-t-il un x tel que $\text{add}(\text{succ}(0), \text{succ}(\text{succ}(0)), x)$?
 - ★ $\exists \vec{x} \text{ add}(\text{succ}(0), \text{succ}(\text{succ}(0)), x)$
- une clause négative correspond à la négation d'une requête
- **sémantiques** d'un programme P (en) logique :
 - ▶ qu'est ce qu'on peut calculer avec P ?
 - ★ *sémantique déclarative*
 - ▶ comment peut-on calculer avec P (i.e. comment **exécuter** un programme en logique) ?
 - ★ *sémantique opérationnelle*

Sémantique déclarative

- qu'est ce qu'on peut calculer avec un programme P à partir d'une requête R ?
 - ▶ une **solution** θ (i.e. une substitution – une fonction qui donne une valeur aux variables de la requête) telle que

$$P \models \theta(R)$$

★ tous les modèles de P sont des modèles de $\theta(R)$

- exemple

$$\models \left\{ \begin{array}{l} \forall x \text{ add}(0, x, x) \\ \forall x \forall y \forall z \text{ add}(x, y, z) \Rightarrow \text{add}(\text{succ}(x), y, \text{succ}(z)) \end{array} \right\}$$

$$\models \theta(\text{add}(\text{succ}(0), \text{succ}(\text{succ}(0)), x))$$

θ est la substitution qui associe $\text{succ}(\text{succ}(\text{succ}(0)))$ à x

- pourquoi se restreindre aux clauses de Horn ?
 - ▶ tout ensemble fini de clause de Horn admet un plus petit modèle

Sémantique opérationnelle

programme P	requête R
$\{\forall \vec{X} ((A_1 \wedge \dots \wedge A_n) \Rightarrow A)\}$	$\exists \vec{X} (B_1 \wedge \dots \wedge B_k)$

exécution : soumission d'une requête R étant donné un programme P

- construction d'une preuve de $\exists \vec{X} (B_1 \wedge \dots \wedge B_k)$ à partir de P par réfutation
 - ▶ réfutation d'un ensemble E de formules logiques : preuve de **false** à partir de E
- réfutation de l'ensemble de clauses $P \cup \neg \exists \vec{X} (B_1 \wedge \dots \wedge B_k)$
 - ▶ en utilisant la **SLD-résolution** (règle de déduction)
- extraction de cette preuve des valeurs associées à \vec{X}
 - ▶ il s'agit d'une substitution θ appelée **réponse** de P à R

SLD-résolution

● Résolution

$$\frac{\neg A_1 \vee \dots \vee \neg A_{n_1} \vee B_1 \vee \dots \vee B_j \vee \dots \vee B_{m_1} \quad \neg A'_1 \vee \dots \vee \neg A'_i \vee \dots \vee \neg A'_{n_2} \vee B'_1 \vee \dots \vee B'_{m_2}}{\theta \left(\begin{array}{l} \neg A_1 \vee \dots \vee \neg A_{n_1} \vee \neg A'_1 \vee \dots \vee \neg A'_{i-1} \vee \neg A'_{i+1} \vee \dots \vee \neg A'_{n_2} \\ \vee \quad B_1 \vee \dots \vee B_{j-1} \vee B_{j+1} \vee \dots \vee B_{m_1} \vee B'_1 \vee \dots \vee B'_{m_2} \end{array} \right)}$$

si θ est un **unificateur** des atomes A'_i et B_j (i.e. $\theta(A'_i) = \theta(B_j)$).

- ▶ on choisit l'unificateur le plus général (*mgu* : *most general unifier*)

● SLD-résolution (*Selection Linear Definite Resolution*)

$$\frac{\begin{array}{l} \neg A_1 \vee \dots \vee \neg A_{n_1} \vee B \quad (\text{clause définie du programme}) \\ \neg A'_1 \vee \dots \vee \neg A'_i \vee \dots \vee \neg A'_{n_2} \quad (\text{clause négative (requête)}) \end{array}}{\underbrace{\theta(\neg A'_1 \vee \dots \vee \neg A'_{i-1} \vee \neg A_1 \vee \dots \vee \neg A_{n_1} \vee \neg A'_{i+1} \vee \dots \vee \neg A'_{n_2})}_{\text{clause négative}}} \quad (\theta = mgu(B, A'_i))$$

● SLD-dérivation

$$\begin{array}{ccccccc} C_0 & & C_1 & & & & C_n \\ \searrow & & \searrow & & & & \searrow \\ R_0 \xrightarrow{\theta_0} & R_1 \xrightarrow{\theta_1} & \dots & R_n \xrightarrow{\theta_n} \emptyset \end{array}$$

réponse : composition $\theta = \theta_n \circ \dots \circ \theta_1$

Exemple

$$P = \left\{ \begin{array}{l} \forall x \text{ add}(0, x, x) \\ \forall x \forall y \forall z \text{ add}(x, y, z) \Rightarrow \text{add}(\text{succ}(x), y, \text{succ}(z)) \end{array} \right\}$$

$$\neg \text{add}(x_1, y_1, z_1) \vee \text{add}(\text{succ}(x_1), y_1, \text{succ}(z_1))$$

$$\theta_0 = \begin{bmatrix} x_1 & y_1 & x \\ 0 & \text{succ}(\text{succ}(0)) & \text{succ}(z_1) \end{bmatrix}$$

$$\text{add}(0, x_2, x_2)$$

$$\theta_1 = \begin{bmatrix} x_2 & z_1 \\ \text{succ}(\text{succ}(0)) & \text{succ}(\text{succ}(0)) \end{bmatrix}$$

$$\neg \text{add}(\text{succ}(0), \text{succ}(\text{succ}(0)), x)$$

$$\downarrow$$

$$\neg \text{add}(0, \text{succ}(\text{succ}(0)), z_1)$$

$$\downarrow$$

$$\emptyset$$

$$\theta_1 \circ \theta_0(x) = \text{succ}(\text{succ}(\text{succ}(0)))$$

Etude d'un langage de programmation : fragment Hornien de la logique

- de la *syntaxe* : clauses
- de la *sémantique* : solutions (modèles) et réponses (preuves)
- des *propriétés* :
 - ▶ *soundness* : si θ est une réponse alors θ est une solution.
 - ▶ *completeness* : toute solution est une « instance » d'une réponse
 - ▶ *lifting lemma* : si $\theta(R)$ admet une réponse alors R admet une réponse.
 - ▶ *non-déterminisme* :
 - ★ dans le choix de la clause
 - ★ dans le choix de l'atome considéré dans la clause négative

Non-déterminisme

- **choix de la clause** : *non-déterminisme par ignorance (Don't know)*
mécanisme de *backtrack*
 - ▶ stratégie PROLOG : ordre d'apparition des clauses dans le programme
- **choix de l'atome** : *non-déterminisme par indifférence (Don't care)*
 - ▶ **lemme de commutation** (*Switching lemma*)

$$\begin{array}{c}
 \underbrace{(\dots, L_1, \dots, L_2, \dots)} \\
 \begin{array}{ccc}
 & R_0 & \\
 C_1 \swarrow & & \searrow C_2 \\
 \theta(\dots, C_1^-, \dots, L_2, \dots) & & \sigma(\dots, L_1, \dots, C_2^-, \dots) \\
 C_2 \downarrow & & \downarrow C_1 \\
 \eta\theta(\dots, C_1^-, \dots, C_2^-, \dots) \approx \mu\sigma(\dots, C_1^-, \dots, C_2^-, \dots)
 \end{array}
 \end{array}$$

- ▶ stratégie PROLOG : choix l'atome le plus à gauche