Sémantique d'un petit langage impératif

Deuxième cours: IMP *toujours*

- IMP: $X, Y, \ldots$ expressions arithmétiques, booléennes
  $:=$; `if ... then ... else ...` `skip` `while ... do ...`
- sémantique opérationnelle évaluation $c, \sigma \mapsto \texttt{skip}, \sigma'$
  transition: $c, \sigma \to c', \sigma'$
- lexique
  - jugement (généralement notation chic pour un tuple)
  - règle d'inférence
    $$\frac{\text{prémisse}_1 \quad \ldots \quad \text{prémisse}_k}{\text{conclusion}} \text{ side condition}$$
    - axiome: pas de prémisse
    - les (méta)variables sont implicitement quantifiées universellement
  - dérivation: arbre (pour une fois à l'endroit) construit avec les règles d'inférence
- configuration σ: associe un entier à chaque variable (tuple, fonction) Σ ensemble des configurations

Sémantique dénotationnelle de IMP*: méthode*

- on ne paraphrase plus, on *envoie* (dans une structure mathématique)
  - envoyer = "calculer la dénotation"
  - si c est une commande (un programme) IMP, on note $[\![c]\!]$ la dénotation de c
    (et on surcharge la notation: $[\![X > 32]\!]$ pour représenter la dénotation de l'expression booléenne $X > 32$)
- on envoie de manière *compositionnelle*
  la sémantique de la construction $C(c_1, \ldots, c_k)$, notée $[\![C(c_1, \ldots, c_k)]\!]$, est définie à partir de $[\![c_1]\!], \ldots, [\![c_k]\!]$
- esquissons:
  - on envoie où?
  - expliquez-moi la sémantique de
    $:=$, `skip`, `if ... then ... else ...`, ;
  - le clou du spectacle: `while`
    - `while` est une "moulinette" sur les commandes
    - il faut se promener dans l'espace où l'on dénote

Domaines

théorie des domaines (Scott, Plotkin)

petit interlude

Retour à la sémantique dénotationnelle de IMP

- on construit la dénotation dans le domaine $D = \Sigma \to \Sigma_\perp$ (ordonné point à point)
- $[\![\texttt{if } b \texttt{ then } c \texttt{ else } c']\!] \stackrel{\text{def}}{=} \lambda\sigma.\textbf{if } [\![b]\!]\sigma \textbf{ then } [\![c]\!]\sigma \textbf{ else } [\![c']\!]\sigma$
  - λ, **if**: constructions au niveau meta
    if true **then** x **else** $y = x$, **if** false **then** x **else** $y = y$)
  - on remarque que le calcul de $[\![\texttt{if true then } c \texttt{ else } c']\!]$ a pour effet d'éliminer le "code mort" $[\![c']\!]$
- $[\![X := a]\!] \stackrel{\text{def}}{=} \lambda\sigma.\, \sigma_{\{X \leftarrow [\![a]\!]\sigma\}}$
- $[\![\texttt{skip}]\!] \stackrel{\text{def}}{=} \lambda\sigma.\sigma$
- $[\![c; c']\!] \stackrel{\text{def}}{=} \lambda\sigma.[\![c']\!]_{\perp\!\!\perp}([\![c]\!]\sigma) \quad = [\![c']\!]_{\perp\!\!\perp} \circ [\![c]\!]$ noté $[\![c']\!] \circ [\![c]\!]$

ne soyons pas pédants

Sémantique du `while`

pour calculer $[\![\texttt{while}\ b\ \texttt{do}\ c]\!]$,

- on part de l'équation
  $\texttt{while}\ b\ \texttt{do}\ c \quad = \quad \texttt{if}\ b\ \texttt{then}\ (c;\texttt{while}\ b\ \texttt{do}\ c)\ \texttt{else}\ \texttt{skip}$
- on considère $f : [D \to D]$ définie par $(D = \Sigma \to \Sigma_\perp)$
  $$f(C) \stackrel{\text{def}}{=} \lambda\sigma.\ \textbf{if}\ [\![b]\!]\sigma\ \textbf{then}\ C([\![c]\!]\sigma)\ \textbf{else}\ [\![\texttt{skip}]\!]\sigma$$
  on cherche une solution de $C = f(C)$
- le théorème du point fixe suggère μf
  au passage: on privilégie le *plus petit* point fixe

  et donc
  $$[\![\texttt{while}\ b\ \texttt{do}\ c]\!] \stackrel{\text{def}}{=} \mu\big(\lambda C.\ \lambda\sigma.\ \textbf{if}\ [\![b]\!]\sigma\ \textbf{then}\ C([\![c]\!]\sigma)\ \textbf{else}\ [\![\texttt{skip}]\!]\sigma\big)$$
- au fait, c'est bien une fonction continue, comme toutes les $[\![c]\!]$ que l'on a définies

Ce qui se passe dans la sémantique du `while`

- considérons le programme
  $\mathsf{w} \stackrel{\text{def}}{=} \texttt{while}\ X > 0\ \texttt{do}\ (Y{:=}X * Y; X{:=}X - 1)$
- on se restreint à $\mathit{vars} = \{X, Y\}$, donc une configuration est donnée par un couple (x, y)
- il faut résoudre $d = f(d)$ avec
  $$f(d)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ d(x-1, x*y) & \text{si } x > 0 \end{cases}$$
- pour trouver le plus petit point fixe de $\boldsymbol{\lambda} d.\, f(d)$, décrivons la chaîne $\bot, f(\bot), f^2(\bot), \ldots$

Exemple avec `while`, *suite*

$$f(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ \bot & \text{si } x \geq 1 \end{cases} \qquad f^2(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ (0,y) & \text{si } x = 1 \\ \bot & \text{si } x \geq 2 \end{cases}$$

$$f^3(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ (0,y) & \text{si } x = 1 \\ (0,2*y) & \text{si } x = 2 \\ \bot & \text{si } x \geq 3 \end{cases} \qquad f^4(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ (0,y) & \text{si } x = 1 \\ (0,2*y) & \text{si } x = 2 \\ (0,6*y) & \text{si } x = 3 \\ \bot & \text{si } x \geq 4 \end{cases}$$

$$f^n(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ (0,(x!)*y) & \text{si } 0 < x < n \\ \bot & \text{si } x \geq n \end{cases}$$

$$f^\infty(\bot)(x,y) = \begin{cases} (x,y) & \text{si } x \leq 0 \\ (0,(x!)*y) & \text{si } x > 0 \end{cases}$$

- on peut vérifier que $f(f^\infty)(x,y) = f^\infty(x,y)$

  c'est plus rigolo que les entiers couchés ou debout

- s'agissant de `while` true `do` `skip`, l'équation s'écrit

  $C = \lambda\sigma.\,\textbf{if } [\![\text{true}]\!]\sigma \textbf{ then } C([\![\texttt{skip}]\!]\sigma) \textbf{ else } [\![\texttt{skip}]\!]\sigma$, soit

  $C = \lambda\sigma.\,C\sigma$

Contemplons la sémantique dénotationnelle

- on a une sémantique compositionnelle
- qui abstrait des particularités du langage
  - on ne veut pas savoir s'il y a un `while` ou un `repeat`
  - les programmes $X{:=}X+1;X{:=}X+1$ et $X{:=}X+2$ sont identifiés
  - idem pour $X{:=}X+1;Y{:=}Y*2$ et $Y{:=}Y*2;X{:=}X+1$
- **Théorème (coïncidence entre les sémantiques):**
  $c,\sigma \mapsto \texttt{skip},\sigma'$ ssi $[\![c]\!]\sigma = \sigma'$.
  preuve: au tableau
- on remarque que le choix du plus petit point fixe pour `while` 'est le bon'
- plusieurs visions d'un même programme: dénotation, sens,...

Au sujet de l'induction

- Romain vous a expliqué en TD l'induction par règles:
  - un ensemble $\mathcal{E}$
  - un ensemble d'axiomes $\mathcal{A}$ et de règles $\mathcal{R} \subseteq \mathcal{P}_{\text{fin}}(\mathcal{E}) \times \mathcal{E}$
  - $\mathcal{A}, \mathcal{R}$ définissent le sous-ensemble (de $\mathcal{E}$) E des racines de tous les arbres de dérivation valides construits à l'aide de $\mathcal{A}, \mathcal{R}$
  - on va dire que c'est juste $\mathcal{R}$, les éléments de $\mathcal{A}$ étant vus comme des règles à zéro prémisse
- E est le plus petit sous-ensemble de $\mathcal{E}$ clos par $\mathcal{R}$
  - E est clos par $\mathcal{R}$ (par définition)
  - si Q est clos par $\mathcal{R}$, Q contient E
- une autre vision de la construction de E: théorème de Knaster-Tarski

Le théorème de Knaster-Tarski

- un *treillis complet* est un ordre partiel dans lequel **toute** partie a un plus petit majorant ($\bigsqcup$) et un plus grand minorant ($\bigsqcap$)
- c'est le cas pour l'ensemble des parties de $\mathcal{E}$, avec l'ordre $\subseteq$
- **Théorème:** f fonction croissante sur un treillis complet. On pose $m \stackrel{\text{def}}{=} \bigsqcap\{x.\, f(x) \sqsubseteq x\}$.
  Alors m est un point fixe de f et c'est le plus petit pré-point fixe ($f(m) \sqsubseteq m$) de f.
  - preuve: au tableau
  - cela nous donne le principe du raisonnement par induction: si on montre que $f(d) \sqsubseteq d$, on déduit $m \sqsubseteq d$
  - p.ex. pour les listes, si une propriété vaut pour `nil` et est préservée par `cons`, alors elle vaut pour toutes les listes (finies)
- mise à l'envers ($\sqsupseteq, \bigsqcup$): c'est une autre histoire

Retour à $\mathcal{E}$,$\mathcal{R}$ et $\mathcal{F}$

- on considère l'opérateur $\mathcal{F}$ sur les parties de $\mathcal{E}$ défini par
  $\mathcal{F}(P) \stackrel{\text{def}}{=} \{y \in \mathcal{E}. \exists X \subseteq P. (X, y) \in \mathcal{R}\}$
  - $\mathcal{F}$ est croissant
  - $P \subseteq \mathcal{E}$ est clos par $\mathcal{R}$ ssi $\mathcal{F}(P) \subseteq P$, i.e. P pré-point fixe de $\mathcal{F}$
- $E = \bigcap\{P. \mathcal{F}(P) \subseteq P\}$
  on retrouve E, le plus petit ensemble clos par les règles de $\mathcal{R}$
- principe de l'induction par les règles:
  de $\mathcal{F}(P) \subseteq P$, on déduit $E \subseteq P$
- E peut aussi être obtenu comme $\bigsqcup_n \mathcal{F}^n(\emptyset)$

en Coq

- Coq implémente CCI, le calcul des constructions *inductives*
- mécanisme uniforme pour la définition de types inductifs (structures de données, prédicats)
- à une telle définition sont associés des schémas d'élimination permettant de raisonner de manière inductive
- exemple: fichier `rule_induction.v`