Un petit langage impératif, IMP.

1 Syntaxe et sémantique opérationnelle.

On se donne $\mathbb{Z}$ et V un ensemble infini de variables IMP, notées x, y, z. On définit plusieurs grammaires :

Arith. Les expressions arithmétiques $a ::= \underline{k} \mid a_1 \oplus a_2 \mid x$;[1]

Valeurs booléennes. $bv ::= \texttt{true} \mid \texttt{false}$;

Bool. Les expressions booléennes $b ::= bv \mid b_1 \wedge b_2 \mid a_1 \geq a_2$;

Com. Les commandes $c ::= x \texttt{ := } a \mid c_1 \texttt{ ; } c_2 \mid \texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2 \mid \texttt{while } b \texttt{ do } c \mid \texttt{skip}$.

Sans explicitement le dire, on s'autorise à étendre les expressions arithmétiques avec, par exemple, les produits, les soustractions. De même pour les expressions booléennes.

On définit, par induction sur c, $\mathsf{Vars}(c)$ l'ensemble des variables dans la commande c. Il y a 5 cas.

Exemple 1. La commande

$$z \texttt{ := } 1 \texttt{ ; while } (x > 0) \texttt{ do } (z \texttt{ := } z \times x \texttt{ ; } x \texttt{ := } x - 1)$$

représente un programme calculant la factorielle d'un nombre x. On le notera c_{fact}.

1. Et on arrêtera rapidement de mettre des barres sous les entiers et d'entourer les plus.

1.1 Sémantique opérationnelle à grands pas.

Définition 1 (États mémoire)**.** On se donne $\mathscr{M}$ un ensemble de dictionnaires, notés σ, σ', *etc* sur $(\mathrm{V}, \mathbb{Z})$.

Si $x \in \mathrm{dom}(\sigma)$ et $k \in \mathbb{Z}$ on note $\sigma[x \mapsto k]$ l'état mémoire σ' défini par

- ▷ $\sigma'(x) := k$;
- ▷ $\sigma'(y) := \sigma(y)$ si $y \in \mathrm{dom}(\sigma) \setminus \{x\}$.

Ici, on *écrase* la valeur de x dans l'état mémoire σ.

On définit $c, \sigma \Downarrow \sigma'$ (l'évaluation de c sur σ produit σ', c fait passer de σ à σ') par les règles d'inférences ci-dessous

$$\frac{}{\texttt{skip}, \sigma \Downarrow \sigma}\ \mathscr{E}_{\text{skip}} \qquad \frac{c_1, \sigma \Downarrow \sigma' \qquad c_2, \sigma' \Downarrow \sigma''}{c_1 \ \texttt{;}\ c_2, \sigma \Downarrow \sigma''}\ \mathscr{E}_{\text{seq}}$$

$$\frac{b, \sigma \Downarrow \texttt{true} \qquad c_1, \sigma \Downarrow \sigma'}{\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma \Downarrow \sigma'}\ \mathscr{E}_{\text{it}} \qquad \frac{b, \sigma \Downarrow \texttt{false} \qquad c_2, \sigma \Downarrow \sigma'}{\texttt{if}\ b\ \texttt{then}\ c_1\ \texttt{else}\ c_2, \sigma \Downarrow \sigma'}\ \mathscr{E}_{\text{if}}$$

$$\sigma' = \sigma[x \mapsto k]\ \frac{a, \sigma \Downarrow k}{x\ \texttt{:=}\ a, \sigma \Downarrow \sigma'}\ \mathscr{E}_{\text{aff}} \qquad \frac{b, \sigma \Downarrow \texttt{false}}{\texttt{while}\ b\ \texttt{do}\ c, \sigma \Downarrow \sigma}\ \mathscr{E}_{\text{wf}}$$

$$\frac{b, \sigma \Downarrow \texttt{true} \qquad c, \sigma \Downarrow \sigma' \qquad \texttt{while}\ b\ \texttt{do}\ c, \sigma' \Downarrow \sigma''}{\texttt{while}\ b\ \texttt{do}\ c, \sigma \Downarrow \sigma''}\ \mathscr{E}_{\text{wf}}$$

où l'on a deux autres relations (la couleur a de l'importance ici) :

- ▷ *l'évaluation des expressions arithmétiques* $a, \sigma \Downarrow k$ (a s'*évalue* en k dans σ)

$$\frac{}{\underline{k}, \sigma \Downarrow k} \qquad \sigma(x) = k\ \frac{}{x, \sigma \Downarrow k} \qquad k = k_1 + k_2\ \frac{a_1, \sigma \Downarrow k_1 \qquad a_2, \sigma \Downarrow k_2}{a_1 \oplus a_2, \sigma \Downarrow k}$$

- *l'évaluation des expressions booléennes* $b, \sigma \Downarrow bv$ (b s'*évalue* en bv dans σ)

$$\frac{}{bv, \sigma \Downarrow bv} \qquad {\scriptstyle bv = \mathtt{true} \text{ ssi } bv_1 \text{ et } bv_2} \frac{b_1, \sigma \Downarrow bv_1 \qquad b_2, \sigma \Downarrow bv_2}{b_1 \wedge b_2, \sigma \Downarrow bv}$$

$$ {\scriptstyle bv = \mathtt{true} \text{ ssi } k_1 \geq k_2} \frac{a_1, \sigma \Downarrow k_1 \qquad a_2, \sigma \Downarrow k_2}{a_1 \geq a_2, \sigma \Downarrow bv.}$$

Remarque 1 (des « variables » partout !).

- Les variables dans FUN sont les paramètres des fonctions, elles peuvent être liées, libres, et on peut procéder à de l'α-conversion. [2]
- Les variables d'unification sont des inconnues. Il y a une notion de substitution, mais pas de liaison.
- Les variables dans IMP sont des cases mémoire, des registres, et il n'y a pas de liaison.

Remarque 2. Soit c une commande, et $\sigma \in \mathscr{M}$. Il peut arriver que, quel que soit $\sigma' \in \mathscr{M}$, on n'ait pas $c, \sigma \Downarrow \sigma'$, soit parce que $\mathrm{dom}(\sigma)$ est trop petit, et l'exécution se bloque ; soit parce que le programme diverge, par exemple

$$\texttt{while true do skip}$$

diverge car on n'a pas de dérivation finies :

$$\frac{\mathtt{true}, \sigma \Downarrow \mathtt{true} \qquad \dfrac{}{\mathtt{skip}, \sigma \Downarrow \sigma} \qquad \texttt{while true do skip}, \sigma \Downarrow ?}{\texttt{while true do skip}, \sigma \Downarrow ?} \mathscr{E}_{\mathrm{wt}}.$$

On peut définir des petits pas pour IMP (vu plus tard en cours, ou en TD), mais on s'intéresse plus à une autre sémantique, la *sémantique dénotationnelle*.

2. C'est similaire au cas de la variable x dans $\int_0^7 f(x)\,\mathrm{d}x$.

2 Sémantique dénotationnelle de IMP.

$$c \xrightarrow{\mathscr{D}} \text{fonction partielle } \mathscr{M} \rightharpoonup \mathscr{M}$$

$$\updownarrow$$

$$\text{relation binaire sur } \mathscr{M} \text{ déterministe/fonctionnelle}$$

On définit les relations

- ▷ $\mathscr{D}(a) \subseteq \mathscr{M} \times \mathbb{Z}$ fonctionnelle ;
- ▷ $\mathscr{D}(b) \subseteq \mathscr{M} \times \{\texttt{true}, \texttt{false}\}$ fonctionnelle ;
- ▷ $\mathscr{D}(c) \subseteq \mathscr{M} \times \mathscr{M}$ fonctionnelle.

On ne traitera que la définition de $\mathscr{D}(c)$, les autres sont laissées en exercice.

On définit $\mathscr{D}(c)$ par induction sur c, il y a 5 cas.

- ▷ $\mathscr{D}(\texttt{skip}) = \{(\sigma, \sigma)\}$;
- ▷ $\mathscr{D}(x \texttt{ := } a) = \{(\sigma, \sigma') \mid x \in \operatorname{dom}(\sigma), \sigma' = \sigma[x \mapsto k] \text{ et } (\sigma, k) \in \mathscr{D}(a)\}$;
- ▷ $\mathscr{D}(\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2) = \{(\sigma, \sigma' \mid (\sigma, \texttt{true}) \in \mathscr{D}(b), (\sigma, \sigma') \in \mathscr{D}(c_1))\} \cup$ $\{(\sigma, \sigma' \mid (\sigma, \texttt{false}) \in \mathscr{D}(b), (\sigma, \sigma') \in \mathscr{D}(c_2))\}$;
- ▷ $\mathscr{D}(c_1 \texttt{ := } c_2) = \{(\sigma, \sigma'') \mid \exists \sigma', (\sigma, \sigma') \in \mathscr{D}(c_1) \text{ et } (\sigma', \sigma'') \in \mathscr{D}(c_2)\}$;[3]
- ▷ $\mathscr{D}(\texttt{while } b \texttt{ do } c) = ???$.

Pour la sémantique dénotationnelle de la boucle `while`, on s'appuie sur l'« équivalence » des commandes

$$\texttt{while } b \texttt{ do } c \qquad \text{et} \qquad \texttt{if } b \texttt{ then } (c \texttt{ := while } b \texttt{ do } c) \texttt{ else skip}.$$

On introduit, pour $\mathsf{R} \subseteq \mathscr{M} \times \mathscr{M}$, la relation

$$\begin{aligned} F(\mathsf{R}) = \quad & \{(\sigma, \sigma) \mid (\sigma, \texttt{false}) \in \mathscr{D}(b)\} \\ & \cup \{(\sigma, \sigma') \mid (\sigma, \texttt{true}) \in \mathscr{D}(b), \exists \sigma', (\sigma, \sigma') \in \mathscr{D}(c) \text{ et } (\sigma', \sigma'') \in \mathsf{R}\}. \end{aligned}$$

On a envie de définir $\mathscr{D}(\texttt{while } b \texttt{ do } c)$ comme un point fixe de F.

3. C'est la composée de $\mathscr{D}(c_2)$ avec $\mathscr{D}(c_1)$.

L'ensemble des relations binaires fonctionnelles sur $\mathscr{M}$ **n'est pas** un treillis complet (à cause se $\mathsf{R}_1 \cup \mathsf{R}_2$ qui n'est pas nécessairement fonctionnelle). On ne peut donc pas appliquer le théorème de Knaster-Tarski.

En revanche, c'est un domaine : si $e_0 \subseteq e_1 \subseteq \cdots \subseteq e_n \subseteq \cdots$ alors l'union $\bigcup_{i\geq 0} e_i$ existe. L'inclusion $e \subseteq e'$ signifie que e' est « plus définie » que e. L'ensemble des relations fonctionnelles sur $\mathscr{M}$ est donc un domaine avec $\bot = \emptyset$. On sait donc que, pour toute fonction F continue, alors F admet un point fixe, qui est égal à

$$\emptyset \cup F(\emptyset) \cup F^2(\emptyset) \cup \cdots = \bigcup_{i\geq 0} F^i(\emptyset).$$

La fonction F définie plus haut est continue, ce qui nous permet de définir

$$\mathscr{D}(\texttt{while}\ b\ \texttt{do}\ c) = \bigcup_{i\geq 0} F^i(\emptyset).$$

Exemple 2. On considère $c_0 = \texttt{while}\ x \neq 3\ \texttt{do}\ x \texttt{ := } x - 1$. Ainsi, la fonction F définie avant $c = c_0$ est

$$\begin{aligned} F_0(\mathsf{R}) = \quad & \{(\sigma,\sigma) \mid \sigma(x) = 3\} \\ & \cup \{(\sigma,\sigma'') \mid \sigma(x) \neq 3, \exists \sigma', \sigma = [x \mapsto \sigma(x) - 1], (\sigma,\sigma') \in \mathsf{R}\}. \end{aligned}$$

On a

- $F_0^0(\emptyset) = \{(\sigma,\sigma) \mid \sigma(x) = 3\}$;
- $F_0^1(\emptyset) = \{(\sigma,\sigma) \mid \sigma(x) = 3\} \cup \{(\sigma,\sigma') \mid \sigma' = [x \mapsto 3], \sigma(x) = 4\}$;
- $F_0^2(\emptyset) = \{(\sigma,\sigma) \mid \sigma(x) = 3\} \cup \{(\sigma,\sigma') \mid \sigma' = [x \mapsto 3], \sigma(x) \in \{4,5\}\}$;
- $F_0^2(\emptyset) = \{(\sigma,\sigma) \mid \sigma(x) = 3\} \cup \{(\sigma,\sigma') \mid \sigma' = [x \mapsto 3], \sigma(x) \in \{4,5,6\}\}$;
- *etc.*

On a bien

$$\emptyset \subseteq F_0(\emptyset) \subseteq F_0^2(\emptyset) \subseteq \cdots.$$

Si $\sigma(x) = 0$, alors quel que soit σ', on a $(\sigma,\sigma') \notin \mathscr{D}(c_0)$.

Exemple 3. Ainsi défini,

$$\mathscr{D}(\texttt{while true do skip}) = \emptyset.$$

Théorème 1. On a $c, \sigma \Downarrow \sigma'$ si et seulement si $(\sigma, \sigma') \in \mathscr{D}(c)$.

Preuve.
- ▷ « $\implies$ » Par induction sur la relation $c, \sigma \Downarrow \sigma'$.
- ▷ « $\impliedby$ » Par induction sur c, où l'on utilise le résultat suivant :

$$\forall n, \quad (\sigma, \sigma') \in F^n(\emptyset) \implies c, \sigma \Downarrow \sigma'.$$

□

Lemme 1. Quels que soient c, σ, σ_1, si c, σ, σ_1 alors,

$$\forall \sigma_2, \quad c, \sigma \Downarrow \sigma_2 \implies \sigma_1 = \sigma_2.$$

Preuve. Une mauvaise idée est de procéder par induction sur c. Il y a 5 cas, et dans le cas `while`, ça bloque parce que la relation grands pas n'est pas définie par induction sur c dans le cas `while`.

On procède par induction sur $c, \sigma \Downarrow \sigma_1$. □

De manière générale, avec IMP, on ne montre pas des résultats de la forme $c, \sigma \Downarrow \sigma' \implies \mathscr{P}$ par induction sur c, car cela ne fonctionne pas, on n'a pas les bonnes hypothèses. On procède par induction sur la relation $c, \sigma \Downarrow \sigma'$.

3 Coinduction.

On retourne sur le théorème de Knaster-Tarski pour la définition d'ensembles et de relations. En notant E l'ensemble ambiant, on travaille dans le treillis complet $(\wp(E), \subseteq)$, avec des fonctions f croissantes dans $\wp(E)$. Le théorème de Knaster-Tarski nous donne ainsi le plus petit pré-point fixe de f, que l'on notera μf. Le principe de la preuve par induction est ainsi :

$$\text{si } A \subseteq E \text{ vérifie } f(A) \subseteq A \text{ alors on a } \mu f \subseteq A.$$

De plus, si f est continue (car $(\wp(E), \subseteq)$ est un domaine), alors on peut calculer explicitement ce plus petit (pré)-point fixe avec la formule $\bigcup_{n\in\mathbb{N}} f^n(\emptyset)$. On part du « bas » et on ajoute des éléments un par un.

Exemple 4. Pour l'exemple de `nat`, on a

$$\forall A \subseteq E, \qquad f(A) = \{\mathtt{0}\} \cup \{\mathtt{S}\ x \mid x \in A\},$$

c'est une fonction continue, et on a

$$\mu f = \{\mathtt{S}^n\mathtt{0} \mid n \in \mathbb{N}\},$$

avec $\mathtt{S}^n\ x = \mathtt{S}\ \mathtt{S}\ \cdots\ \mathtt{S}\ x$ et la convention $\mathtt{S}^0\ x = x$. En effet, on a l'appartenance de $\mathtt{S}^n\mathtt{0} \in \bigcup_{m\in\mathbb{N}} f^m(\emptyset)$ et $f(\{\mathtt{S}^n\ \mathtt{0}\}) = \{\mathtt{S}^{n+1}\ \mathtt{0}\}$.

Remarque 3 (Remarque fondamentale !)**.** Considérons un treillis complet $(E, \sqsubseteq)$. Alors, le treillis $(E, \sqsupseteq)$ est complet, où l'on note $y \sqsupseteq x$ dès lors que $x \sqsubseteq y$ (on renverse l'ordre).

Un majorant pour $\sqsubseteq$ est un minorant pour $\sqsupseteq$ et inversement. Ainsi, le plus plus petit des majorants $\bigsqcup_{\sqsubseteq} A$ pour $\sqsubseteq$ est le plus petit des minorants $\bigsqcap_{\sqsupseteq} A$ pour $\sqsupseteq$. Réciproquement, le plus petit des majorants pour $\sqsubseteq$, $\bigsqcap_{\sqsubseteq} A$ est égal au plus grand majorant pour la relation $\sqsupseteq$, $\bigsqcup_{\sqsupseteq} A$.

On se place ainsi sur le treillis complet $(\wp(E), \supseteq)$. Une fonction est croissante pour $\subseteq$ si et seulement si elle est croissante pour $\supseteq$ (***attention***, elle n'est pas décroissance pour cette deuxième relation). Appliquons le théorème de Knaster-Tarski sur ce nouveau treillis complet à une fonction croissante. Le théorème nous fournis un pré-point fixe pour l'ordre $\supseteq$ (*i.e.* qui vérifie $f(A) \supseteq A$), c'est-à-dire un post-point fixe pour l'ordre $\subseteq$ (*i.e.* qui vérifie $A \subseteq f(A)$). Et, c'est le plus petit point fixe pour $\supseteq$, donc le plus grand point fixe pour $\subseteq$, que l'on notera νf.

Avec le théorème de point fixe sur les domaines, et en supposant f continue, on calcule explicitement que le plus grand point fixe νf vaut l'intersection $\bigcap_{n\in\mathbb{N}} f^n(E)$. On part du haut, et on nettoie progressivement, on raffine notre partie de E.

Ce que l'on a fait là, cela s'appelle de la ***coinduction***.

Exemple 5. Par exemple, on définit `conat` par coinduction. En Rocq, cela donne le code ci-dessous.

```
CoInductive conat : Set := c0 | cS (n : conat).
```

Code 1 | *Définition de* `conat`

Pour illustrer le « nettoyage » effectué dans la définition coinductive, on considère une feuille étiquetée par le mot « `banane` ». A-t-on `cS banane` $\in$ `conat` ? Premièrement, on a `cS banane` $\in E$ car E est l'ensemble (très grand) des arbres étiquetés par des chaînes de caractères. Deuxièmeement, on a `cS banane` $\in f(E)$ car c'est le successeur de `banane` $\in E$. Troisièmement, et c'est là où ça casse, on a `cS banane` $\notin f^2(E)$ parce que `banane` $\notin f(E)$.

Avec la fonction f définie précédemment, on a

$$f^n(E) = \{\mathtt{c0}, \mathtt{cS}\ \mathtt{c0}, \dots, \mathtt{cS}^{n-1}\ \mathtt{c0}\} \cup \{\mathtt{cS}^n\ x \mid x \in E\}.$$

Ainsi, on récupère tous les entiers de `nat`, mais d'autres entiers (oui, il y en a plusieurs) infinis, ayant ainsi une dérivation infinie. Par exemple, il existe $\omega \in$ `conat` tel que $\omega =$ `cS` ω. En Rocq, pour le définir, on ferai :

```
CoFixpoint ω := cS ω
```

Code 2 | *Définition de ω, un entier infini*

> Pour montrer que $\omega \in \mathsf{conat}$, il faut et il suffit de montrer l'inclusion $\{\omega\} \subseteq f(\{\omega\}) = \{\mathtt{c0}, \mathtt{cS}\ \omega\} = \{\mathtt{c0}, \omega\}$, qui est vraie, et on a ainsi $\{\omega\} \subseteq \mathsf{conat}$.

Le principe de la preuve par coinduction permet d'établir qu'un ensemble est contenu dans le plus grand point fixe. Avec le treillis des parties muni de $\subseteq$, cela permet de montrer que $A \subseteq E$ est inclus dans le plus grand post-point fixe de f et, pour cela, il suffit de montrer que $A \subseteq f(A)$, c'est-à-dire que A est un post-point fixe de f. C'est ce que l'on a fait dans l'exemple avec ω.

Par coinduction, on peut par exemple montrer que l'on a, pour tous états mémoire σ, σ',

$$\texttt{while true do skip}, \sigma \Downarrow \sigma'.$$

4 Divergences en IMP.

On donne une définition coinductive de la divergence en IMP, que l'on notera $c, \sigma \Uparrow$ avec les règles

$$\frac{c_1, \sigma \Uparrow}{c_1 \texttt{ ; } c_2, \sigma \Uparrow} \qquad \frac{c_1, \sigma \Downarrow \sigma' \qquad c_2, \sigma' \Uparrow}{c_1 \texttt{ ; } c_2, \sigma \Uparrow} \qquad \frac{b, \sigma \Downarrow \texttt{true} \qquad c_1, \sigma \Uparrow}{\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \Downarrow}$$

$$\frac{b, \sigma \Downarrow \texttt{false} \qquad c_2, \sigma \Uparrow}{\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2, \sigma \Downarrow} \qquad \frac{b, \sigma \Downarrow \texttt{true} \qquad c, \sigma \Uparrow}{\texttt{while } b \texttt{ do } c, \sigma \Uparrow}$$

$$\frac{b, \sigma \Downarrow \texttt{true} \qquad c, \sigma \Downarrow \sigma' \qquad \texttt{while } b \texttt{ do } c, \sigma' \Uparrow}{\texttt{while } b \texttt{ do } c, \sigma \Uparrow}$$

On n'a pas de règle pour la divergence si $b, \sigma \Downarrow \texttt{false}$, car dans ce cas là, on ne peut pas diverger (c'est équivalent à un `skip`).

Le plus grand point fixe ne contient que des dérivations infinies, qui correspondent à des exécutions divergentes d'un programme IMP à partir d'un état mémoire donné. En effet, ceci vient du fait que, si on interprète ces règles comme des règles inductives, la relation obtenue est l'ensemble vide...

5 Logique de Floyd–Hoare.

On considère des formules logiques, des *assertions* (définies formellement ci-après), que l'on notera A, A', B, *etc*. Un triplet de Hoare est de la forme $\{A\}c\{A'\}$ (la notation est inhabituelle pour les triplets, mais c'est une notation commune dans le cas des triplets de Hoare), où l'on nomme A la *précondition* et A' la *postcondition*.

Exemple 6. Les triplets suivants sont des triplets de Hoare :

1. $\{x \geq 1\}y \texttt{ := } x + 2\{x \geq 1 \wedge y \geq 3\}$ qui est une conclusion naturelle ;
2. $\{n \geq 1\}c_{\text{fact}}\{r = n!\}$ où l'on note c_{fact} la commande

   $$x \texttt{ := } n \texttt{ ; } z \texttt{ := } 1 \texttt{ ; } \texttt{while}\,(x > 0)\,\texttt{do}\,(z \texttt{ := } z \times x \texttt{ ; } x \texttt{ := } x - 1)\ ,$$

   qui calcule naturellement la factorielle de n ;
3. $\{x < 0\}c\{\texttt{true}\}$ même s'il ne nous dit rien d'intéressant (tout état mémoire vérifie $\texttt{true}$) ;
4. $\{x < 0\}c\{\texttt{false}\}$ qui diverge dès lors que $x < 0$.

On considère un ensemble $I \ni i$ infini d'*index*, des « inconnues ». On commence par définir les expressions arithmétiques étendues

$$a ::= \underline{k} \mid a_1 \oplus a_2 \mid x \mid i,$$

puis définit les *assertions* par la grammaire ci-dessous :

$$A ::= bv \mid A_1 \vee A_2 \mid A_1 \wedge A_2 \mid a_1 \geq a_2 \mid \exists i, A.$$

On s'autorisera à étendre, implicitement, les opérations réalisées dans les expressions arithmétiques, et les comparaisons effectuées dans les assertions.

On ajoute la liaison d'α-conversion : les assertions $\exists i, x = 3 * i$ et $\exists j, x = 3 * j$ sont α-équivalentes. On note $i\ell(A)$ l'ensemble des index libres de l'assertion A, et on dira que A est *close* dès lors que $i\ell(A) = \emptyset$. On note aussi $A[{}^{k}\!/\!_{i}]$ l'assertion A où $k \in \mathbb{Z}$ remplace $i \in I$.

Définition 2. Considérons A close et $\sigma \in \mathscr{M}$. On définit par induction sur A (4 cas) une relation constituée de couples (σ, A), notés $\sigma \models A$ (« σ satisfait A »), et en notant $\sigma \not\models A$ lorsque (σ, A) n'est pas dans la relation :

- ▷ $\sigma \models \texttt{true}$ $\forall \sigma \in \mathscr{M}$;
- ▷ $\sigma \models A_1 \vee A_2$ si et seulement si $\sigma \models A_1$ ou $\sigma \models A_2$;
- ▷ $\sigma \models a_1 \geq a_2$ si et seulement si on a $a_1, \sigma \Downarrow k_1$ et $a_2, \sigma \Downarrow k_2$ et $k_1 \geq k_2$;
- ▷ $\sigma \models \exists i, A$ si et seulement s'il existe $k \in \mathbb{Z}$ tel que $\sigma \models A[{}^k/_i]$.

On écrit $\models A$ (« A est valide ») lorsque pour tout σ tel que $\mathrm{dom}(\sigma) \supseteq \mathsf{vars}(A)$, on a $\sigma \models A$.

5.1 Règles de la logique de Hoare : dérivabilité des triplets de Hoare.

Les triplets de Hoare, notés $\{A\}c\{A'\}$ avec A et A' closes, où A est *précondition*, c est commande IMP, et A' est *postcondition*. On définit une relation $\vdash \{A\}c\{A'\}$ sur les triplets de Hoare :

$$\frac{\vdash \{A \wedge b\}c_1\{A'\} \qquad \vdash \{A \wedge \neg b\}c_2\{A'\}}{\vdash \{A\}\texttt{if } b \texttt{ then } c_1 \texttt{ else } c_2\{A'\}} \qquad \frac{}{\vdash \{A\}\texttt{skip}\{A\}}$$

$$\frac{\vdash \{A\}c_1\{A'\} \qquad \vdash \{A'\}c_2\{A''\}}{\vdash \{A\}c_1 \texttt{ ; } c_2\{A''\}} \qquad \frac{\vdash \{A \wedge b\}c\{A\}}{\{A\}\texttt{while } b \texttt{ do } c\{A \wedge \neg B\}}$$

$$\begin{matrix}\models B \implies A \\ \models A' \implies B'\end{matrix}\ \frac{\{A\}c\{A'\}}{\{B\}c\{B'\}} \qquad \frac{}{\{A[{}^a/_x]\}x \texttt{ := } a\{A\}}$$

La dernière règle semble à l'envers, mais c'est parce que la logique de Hoare fonctionne fondamentalement à l'envers.

Dans la règle de dérivation pour la boucle `while`, l'assertion manipulée, A, est un *invariant*.

L'avant dernière règle s'appelle la *règle de conséquence* : on ne manipule pas le programme, la commande, mais plutôt les pré- et post-conditions.

La relation $\vdash \{A\}c\{A'\}$ s'appelle la *sémantique opérationnelle* de IMP.

Définition 3. On définit la relation de *satisfaction*, sur les triplets de la forme $\{A\}c\{A'\}$ avec A, A' closes, avec $\sigma \models \{A\}c\{A'\}$ si et seulement si dès lors que $\sigma \models A$ et $c, \sigma \Downarrow \sigma'$ alors on a $\sigma' \models A'$.

On définit ensuite la relation de *validité* par $\models \{A\}c\{A'\}$ si et seulement si pour tout $\sigma \in \mathscr{M}$, $\sigma \models \{A\}c\{A'\}$.

Théorème 2 (Correction de la logique de Hoare.). Si $\vdash \{A\}c\{A'\}$ alors $\models \{A\}c\{A'\}$.

Preuve. On procède par induction sur $\vdash \{A\}c\{A'\}$. Il y a 6 cas.

- *Règle de conséquence.* On sait

  $$\models B \implies A \text{ et } \models A' \implies B',$$

  et l'hypothèse d'induction. On doit montrer $\models \{B\}c\{B'\}$. Soit σ tel que $\models B$, et supposons $c, \sigma \Downarrow \sigma'$. On a $\models A$ par hypothèse. Puis, par hypothèse d'induction, $\sigma' \models A'$ et donc $\sigma' \models B'$.
- *Règle* `while`. Considérons $c = \texttt{while}\ b\ \texttt{do}\ c_0$. On sait par induction que $\models \{A \wedge b\}c_0\{A\}$ et l'hypothèse d'induction. Il faut montrer $\models \{A\}\texttt{while}\ b\ \texttt{do}\ c_0\{A \wedge \neg b\}$, c'est à dire, si $\sigma \models A$ et $(\star)$: $\texttt{while}\ b\ \texttt{do}\ c_0, \sigma \Downarrow \sigma'$ alors $\sigma' \models A \wedge \neg b$. Pour montrer cela, il est nécessaire de faire une induction sur la dérivation de $(\star)$, « sur le nombre d'itérations dans la boucle ».
- Autres cas en exercice.

$\square$

Le sens inverse, la réciproque, s'appelle la *complétude*. On l'étudiera rapidement après.

Remarque 4. Concrètement, on écrit des programmes annotés.

$$\{x \geq 1\}$$

$$\Downarrow$$

$$\{x \geq 1 \wedge x+2+x+2 \geq 6\}$$

$$y \texttt{ := } x + 2 \texttt{ ;}$$

$$\{x \geq 1 \wedge y + y \geq 6\}$$

$$z \texttt{ := } y + y$$

$$\{x \geq 1 \wedge z \geq 6\}$$

$$\Downarrow$$

$$\{x \geq 1 \wedge z \geq 6\}$$

5.2 Complétude de la logique de Hoare

Pour démontrer la complétude de la logique de Hoare, on s'appuie sur la notion de *plus faible précondition* : étant données une commande c et une assertion B, alors la *plus faible précondition* associée à c, B est l'ensemble des états mémoire

$$\mathrm{wp}(c, B) := \{\sigma \mid c, \sigma \Downarrow \sigma' \implies \sigma' \models B\}.$$

Ainsi, $\mathrm{wp}(c, B)$ est l'ensemble des états mémoire à partir duquels on aboutit à un état satisfaisant B, après une exécution terminante de c.

Proposition 1. Pour toute commande c et toute formule B, il existe une assertion $\mathrm{W}(c, B)$ telle que $\sigma \models \mathrm{W}(c, B)$ si et seulement si $\sigma \in \mathrm{wp}(c, B)$.

Preuve. On procède par induction sur c. Tout fonctionne, sauf pour *while*... Pour le cas de la boucle *while*, on utilise la caractérisation suivante :

$$\sigma \in \mathrm{wp}(\texttt{while } b \texttt{ do } c_0, B)$$

$$\Updownarrow$$

$$\forall k, \forall \sigma_0, \ldots, \sigma_k \text{ si } \sigma_0 = \sigma \text{ et } \forall i < k, (\sigma_i, b \Downarrow \texttt{true} \text{ et } c_0, \sigma_i \Downarrow \sigma_{i+1})$$
$$\text{alors } \sigma_k \models b \vee B.$$

On peut définir cette assertion en définissant des assertions pour :

- décrire un état mémoire σ_i $(X_1^i = v_1 \wedge \cdots \wedge X_n^i = v_n)$;
- exprimer les conditions $\sigma_i, c \Downarrow \sigma_{i+1}$ par induction ;
- exprimer les quantifications $\forall k, \sigma_0, \ldots, \sigma_k$... on demande à Kurt Gödel.

Ainsi, on a bien une assertion $\mathrm{W}(c, B)$ telle que

$$\forall \sigma, \qquad \sigma \in \mathrm{wp}(c, B) \iff \sigma \models \mathrm{W}(c, B).$$

$\square$