UCL/INGI2339

Projet

Unwrap SLIP

18180200 Slinckx Raphael
13829900 Perdaens Antoine

Prof. B. Le Charlier

Table des matières

Définition de la sémantique abstraite

1.1 Les domaines sémantiques abstraits

1.1.1 Valeurs abstraites

Les valeurs sont des références abstraites, des ensembles d'entiers (positifs, négatifs et nuls) et un ensemble spécial *noninit* indiquant une variable pas encore initialisée.

$$\mathbb{V}\text{al}^{\#} = \wp\left(\mathbb{R}\text{ef}^{\#} + \mathbb{V}^{\#}\right) = \wp\left(\mathbb{R}\text{ef}^{\#} + \{null\} + \{noninit\} + \{-, 0, +\}\right)$$
avec $\mathbb{R}\text{ef}^{\#} = \wp(\mathbb{I}\text{nt})$

Pour les valeurs abstraites qui ne sont pas des références, nous définissons la fonction de concrétisation de la manière suivante :

$$\begin{aligned} Cc : \mathbb{V}^{\#} &\rightarrow \wp(\mathbb{V}\text{al}) \\ null &\rightsquigarrow \{null\} \\ noninit &\rightsquigarrow \{noninit\} \\ - &\rightsquigarrow \{i \mid i \in \mathbb{I}\text{nt}, i > 0\} \\ 0 &\rightsquigarrow \{0\} \\ + &\rightsquigarrow \{i \mid i \in \mathbb{I}\text{nt}, i < 0\} \end{aligned}$$

Les références sont un peu plus complexes à concrétiser. Chaque ensemble d'objet concret de même classe (niveau en langage SLIP) est représenté par une instance d'objet abstraite. On a donc un ensemble fini d'objets abstraits chaque objet abstrait représentant toutes les instances possibles d'un objet de même classe.

Les champs des objets abstraits sont l'union des abstractions des champs de tous les objets concrets de même classe instanciés.

$$\begin{aligned} Cc : \mathbb{R}\text{ef}^{\#} &\rightarrow \wp(\mathbb{R}\text{ef}) \\ r^{\#} &\rightsquigarrow \{r \mid \forall s \in \mathbb{S}\text{tore}, \forall r \in dom(s), s(r) = \langle n, \langle v_1, \ldots, v_n \rangle \rangle \wedge \\ &\quad n = r^{\#} \wedge \\ &\quad s \in Cc(s^{\#}) \wedge \\ &\quad \forall i : 1 \leq i \leq n, v_i \in Cc(v_i^{\#}) \wedge \\ &\quad \forall n \in dom(s^{\#}), s^{\#}(n) = \langle n \langle v_1^{\#}, \ldots, v_n^{\#} \rangle \rangle\} \end{aligned}$$

Enfin on définit facilement la concrétisation de $\mathbb{V}al^{\#}$ en utilisant les deux définitions ci-dessus :

$$\begin{aligned} Cc : \mathbb{V}\text{al}^{\#} &\rightarrow \wp(\mathbb{V}\text{al}) \\ v^{\#} &\rightsquigarrow \cup Cc(x), x \in v^{\#} \end{aligned}$$

1.1.2 Environnement abstrait

L'environnement abstrait représente l'ensemble des variables concrètes et leur valeur abstraite.

$$\mathbb{E}\text{nv}^{\#} = (\mathbb{X} + \{this\}) \nrightarrow \mathbb{V}\text{al}^{\#}$$

L'environnement abstrait est simplement définit comme l'abstraction de toutes les valeurs des variable dans l'environnement concret. Les deux environnement ont donc le même domaine.

$$\begin{aligned} Cc : \mathbb{E}\text{nv}^{\#} &\rightarrow \wp(\mathbb{E}\text{nv}) \\ e^{\#} &\rightsquigarrow \{e \mid dom(e) = dom(e^{\#}) \wedge \forall\, x \in dom(e^{\#}), e(x) \in Cc(e^{\#}(x))\} \end{aligned}$$

1.1.3 Instances abstraites

Comme défini plus haut les instances abstraites sont définies comme une classe (le niveau) et un ensemble de taille égale au niveau de valeurs abstraites.

$$\mathbb{I}nst^{\#} = \mathbb{I}nt \times \wp(\mathbb{V}al^{\#})^{*} = \langle n_{\in \mathbb{N}}, \langle v_1, \ldots, v_n \rangle_{\in \wp(\mathbb{V}al^{\#})} \rangle$$

Pour retrouver l'ensemble d'instances concrètes à partir d'une instance abstraite il suffira de prendre l'ensemble des instances qu'il est possible de concrétiser a partir de toutes les combinations de valeurs concrétisées à partir des valeurs abstraites.

$$\begin{aligned} Cc : \mathbb{I}nst^{\#} &\rightarrow \wp(\mathbb{I}nst) \\ i^{\#} &\rightsquigarrow \{\langle n, \langle v_1, \ldots, v_n \rangle \rangle \mid \forall\, i : 1 \leq i \leq n, v_j \in Cc(v_i^{\#}) \wedge \\ &\qquad i^{\#} = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#} \rangle \rangle\} \end{aligned}$$

1.1.4 Store abstrait

Le store abstrait stocke toutes les instances abstraites crées et permet de les retrouver à partir de la référence abstraite.

$$\mathbb{S}tore^{\#} = \mathbb{R}ef^{\#} \nrightarrow \mathbb{I}nst^{\#}$$

$$\begin{aligned} Cc : \mathbb{S}tore^{\#} &\rightarrow \wp(\mathbb{S}tore) \\ s^{\#} &\rightsquigarrow \{s \mid \forall\, r \in dom(s) : s(r) = \langle n, \langle v_1, \ldots, v_n \rangle\rangle, \\ &\quad n \in dom(s^{\#}) \wedge \forall\, i : 1 \leq i \leq n, v_i \in Cc(v_i^{\#}) \wedge \\ &\quad s^{\#}(n) = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#} \rangle\rangle\} \end{aligned}$$

1.1.5 Etat abstrait

L'etat abstrait du programme est similaire à l'état concret. Il faut cependant tenir compte du fait qu'un nombre fini d'états peuvent exister sinon l'interprétation abstraite du programme peut boucler.

A cet effet nous enlevons les ensembles $\mathbb{I}n$ et $\mathbb{O}ut$ de l'état et utilisons les ensembles abstraits au lieu de leur équivalent concret. Seule l'étiquette du point de programme est préservée, puisqu'il existe un nombre fini de ces étiuettes pour un programme donné.

$$\mathbb{E}tat^{\#} = \mathbb{L} \times \mathbb{E}nv^{\#} \times \mathbb{S}tore^{\#} \times \mathbb{P}ile^{\#}$$

$$\begin{aligned} Cc : \mathbb{E}tat^{\#} &\rightarrow \wp(\mathbb{E}tat) \\ \langle l, e^{\#}, s^{\#}, \pi^{\#}_{\in \mathbb{P}ile^{\#}} \rangle &\rightsquigarrow \{\langle l, e, s, \mathbb{I}n, \mathbb{O}ut, \pi \rangle \mid \\ &\quad In, Out \in \mathbb{I}nt^{*} \wedge \\ &\quad e \in Cc(e^{\#}) \wedge e \subset dom(s) \wedge \\ &\quad s \in Cc(s^{\#}) \wedge \\ &\quad \pi \in Cc(\pi^{\#})\} \end{aligned}$$

La pile d'exécution doit aussi être finie il convient donc de faire attention à ce que son contenu soit au maximum un ensemble fini.

Nous allons associer à chaque étiquette de point de retour de méthode un ensemble de tuples. Il s'agit d'un ensemble car une méthode peut retourner à divers endroits dans le programme en fonction de l'endroit ou on l'appelle.

Ces tuples contiendront :
- Une étiquette de retour, indiquant un endroit ou la méthode à été appelée
- Une variable contenant le résultat de la méthode. Le résultat sera évidemment un ensemble abstrait
- Un environnement abstrait tel qu'il était avant l'appel de la méthode à l'endroit ou la méthode retourne

$$\mathbb{P}ile^{\#} = \mathbb{L} \rightharpoonup \wp(\mathbb{L} \times \mathbb{X} \times \mathbb{E}nv^{\#})$$

Pour obtenir les piles concrètes que représente une pile abstraite, il suffira de prendre tous les en-

sembles dans la pile et d'en extraire les tuples d'etiquette, variable de retour et environnement concrétisé.

$$\begin{aligned} Cc : \mathbb{P}ile^{\#} &\rightarrow \wp(\mathbb{P}ile) \\ p^{\#} &\rightsquigarrow \{\langle l \times x \times e\rangle \mid l_m \in dom(\mathbb{P}ile^{\#}) \wedge p^{\#}(l_m) = (\langle l \times x \times e^{\#}\rangle)^* \wedge e \in Cc(e^{\#})\} \end{aligned}$$

1.1.6 Status

Cet ensemble est particulier à l'interprétation abstraite du programme, il indique en tout point de programme si l'instruction est certifiée correcte, incorrecte ou si l'exécution de l'instruction peut éventuellement produire une erreur si certaines conditions sont réunies, ou ne pas en produire dans d'autres cas.

$$\mathbb{S}status = \mathbb{L}abel \rightarrow \{correct, inconnu, incorrect\}$$

Nous définission ici une 'macro' qui permettra de rendre plus concise certaines définitions qui suivent. Elle retourne un status et l'ensemble donné (ou vide) en fonction de la présence ou non de l'element dans cet ensemble.

$$StatusRestrict(set, elt) = \begin{cases} \langle incorrect, \emptyset\rangle & si \quad set \setminus \{elt\} = \emptyset \\ \langle correct, set\rangle & sinon \end{cases}$$

1.2 Les fonctions sémantiques abstraites

1.2.1 Descripteurs

Calcule la valeur d'un descripteur.

$$\mathcal{D}^{\#} : \mathbb{D}es \rightarrow \mathbb{E}nv^{\#} \rightarrow \mathbb{S}tore^{\#} \rightarrow (\mathbb{X} + (\mathbb{R}ef^{\#} \times \mathbb{I}nt) + \{error\})$$

Nous avons donc la variable elle même ou bien la référence abstraite et le numéro du champ demandé dans le cas d'un descripteur de type $x.i$:

$$\begin{aligned} \mathcal{D}^{\#}[\![x]\!]\, e^{\#}\, s^{\#} &= x \\ \mathcal{D}^{\#}[\![x.i]\!]\, e^{\#}\, s^{\#} &= \begin{cases} < r^{\#}, i > & si \quad r_1^{\#} \in e^{\#}(x) \wedge r_1^{\#} \in \mathbb{R}ef^{\#} \wedge \\ & \qquad\quad r^{\#} = \{j : j \leq i \wedge j \in r_1^{\#}\} \\ error & \text{sinon} \end{cases} \end{aligned}$$

Si le descripteur demandé correspond à un champ trop élevé pour l'objet, une erreur est retournée.

1.2.2 Expressions

La valeur des expressions est donnée par la fonction $\mathcal{E}$

$$\mathcal{E}^{\#} : \mathbb{E}xpr \rightarrow \mathbb{E}nv^{\#} \rightarrow \mathbb{S}tore^{\#} \rightarrow (\wp(\mathbb{V}al^{\#}) \times \mathbb{S}tatus)$$

dont voici les valeurs (valeur abstraites et status) simples (sans l'opérateur arithmétique) :

$$
\begin{aligned}
\mathcal{E}^{\#}[\![this]\!]\, e^{\#}\, s^{\#} &= \langle e^{\#}(this), correct\rangle \\
\mathcal{E}^{\#}[\![x]\!]\, e^{\#}\, s^{\#} &= \langle e^{\#}(x), correct\rangle \\
\mathcal{E}^{\#}[\![i]\!]\, e^{\#}\, s^{\#} &= \begin{cases} \langle\{-\}, correct\rangle & si \quad i < 0 \\ \langle\{0\}, correct\rangle & si \quad i = 0 \\ \langle\{+\}, correct\rangle & si \quad i > 0 \end{cases} \\
\mathcal{E}^{\#}[\![x.i]\!]\, e^{\#}\, s^{\#} &= \begin{cases} \langle v^{\#}, status\rangle & si \quad < r^{\#}, i >= \mathcal{D}^{\#}[\![x.i]\!]\, e^{\#}\, s^{\#} \wedge \\ & \quad\; v^{\#}_{*} = x : \forall j \in r^{\#}, s^{\#}(j) = \langle n, \langle v^{\#}_1 ... v^{\#}_n\rangle\rangle \wedge v^{\#}_i \in x \\ & \quad\; status, v^{\#} = StatusRestrict(v^{\#}_{*}, noninit) \\ \langle \emptyset, incorrect\rangle & \text{sinon} \end{cases}
\end{aligned}
$$

Dans le cas de l'expression composée, il faudra définir les opération sur les valeurs abstraites. Voici comment sont définies les opérateurs arithmétiques de base $(+, -, /, *, \%)$ sur les valeurs abstraites (nombre nul, positif, negatif) :

$+$	$+$	0	$-$
$+$	$\{+\}$	$\{+\}$	$\{-,0,+\}$
0	$\{+\}$	$\{0\}$	$\{-\}$
$-$	$\{+,0,-\}$	$\{-\}$	$\{-\}$

$-$	$+$	0	$-$
$+$	$\{+,0,-\}$	$\{+\}$	$\{+\}$
0	$\{-\}$	$\{0\}$	$\{+\}$
$-$	$\{-\}$	$\{-\}$	$\{-,0,+\}$

$/$	$+$	0	$-$
$+$	$\{+,0\}$	$\{error\}$	$\{0,-\}$
0	$\{0\}$	$\{error\}$	$\{0\}$
$-$	$\{0,-\}$	$\{error\}$	$\{0,+\}$

$\%$	$+$	0	$-$
$+$	$\{+,0\}$	$\{error\}$	$\{-,0\}$
0	$\{0\}$	$\{error\}$	$\{0\}$
$-$	$\{+,0\}$	$\{error\}$	$\{-,0\}$

$*$	$+$	0	$-$
$+$	$\{+\}$	$\{0\}$	$\{-\}$
0	$\{0\}$	$\{0\}$	$\{0\}$
$-$	$\{-\}$	$\{0\}$	$\{+\}$

On définit ensuite la fonction abstraite de l'opératuer arithmétique qui renverra un ensemble de valeurs abstraites contenant éventuellement une *error* (par exemple dans le cas d'une addition de deux références, ou d'une division par 0)

$$\mathcal{A}o^{\#} : \mathbb{A}op \rightarrow \mathbb{V}al^{\#} \rightarrow \mathbb{V}al^{\#} \quad \rightarrow \quad \wp(\mathbb{V}al^{\#}) + \{error\}$$

Les opérations se font élément par élément avec ceux dans $v_1^\#$ et $v_2^\#$ en fonction des tableaux ci-dessus. On effectue ensuite l'union des résultats obtenus. Si $v_1^\#$ et/ou $v_2^\#$ contient une $\mathbb{R}ef^\#$, $\{error\}$ est ajouté a l'ensemble des résultats.

Cela permet de définir maintenant la valeur abstraite d'une expression composée, ainsi que le status de l'instruction :

$$\mathcal{E}^\#[\![expr_1\ aop\ \ expr_2]\!]\, e^\#\, s^\# \quad = \quad \begin{cases} \langle v^\#, status\rangle & si \quad status_i \in \{correct, inconnu\} \wedge \\ & \quad v_*^\# = \mathcal{A}o^\#[\![aop]\!]\, v_1^\# v_2^\# e^\# s^\# \wedge \\ & \quad \langle v_i^\#, status_i\rangle = \mathcal{E}^\#[\![expr_i]\!]\, e^\# s^\# \wedge \\ & \quad status, v^\# = StatusRestrict(v_*^\#, error) \\ \langle \emptyset, incorrect\rangle & \text{sinon} \end{cases}$$

L'instruction se soldera par une incertitude si l'ensemble de valeur d'un des deux opérateurs contenait (entre autres) une référence, ou un nombre potentiellement dangereux (contient *error*). Un succes si les deux ensembles contenaient des valeurs légales. Un echec dans les autres cas.

1.2.3 Conditions

Le traitement des branchements conditionnels est un peu particulier, puisqu'il faut produire deux environnements pour chaque résultat possible.

Quand la condition est vérifiée, l'environnement abstrait courant est restreint en tenant compte de la condition et l'exécution continue à la première étiquette. Quand la condition est fausse, l'environnement abstrait courant est restreint en tenant compte de la condition et l'exécution continue à la seconde étiquette. Notons cependant que l'interprétation abstraite se fera en parallèle, dans les deux branches à la fois, même si dans le code celle-ci sera séquentielle.

1.2.3.1 Fonction de restriction

Voici la fonction de restriction de l'environnement. Elle utilise une expression, un ensemble de valeurs abstraites et un environnement et un store initiaux puis renvoie l'environnement restreint en fonction des différents cas d'expression ci-dessous.

$$\mathcal{R}^\# : \mathbb{E}\text{xpr} \to \wp(\mathbb{V}al^\#) \to \mathbb{E}nv^\# \to \mathbb{S}tore^\# \to \mathbb{E}nv^\#$$

Dans le cas d'une variable ou *this*, on effectue simplement l'intersection entre la valeur abstraite de la variable et l'ensemble donné en paramètre pour ne retourner que les valeurs communes aux deux ensembles dans l'environnement retourné.

Dans le cas d'un accès à un champ d'une instance abstraite, on va enlever de l'environnement initial toutes les instances abstraites qui ont un niveau strictement inférieur au numéro du champ demandé, puisque si la condition contient l'evaluation d'une telle expression, la suite du code doit forcément accéder à des objets d'un niveau au moins égal a celui demandé dans la condition.

$$
\begin{aligned}
\mathcal{R}^{\#}[\![x]\!]\, v^{\#}\, e^{\#}\, s^{\#} &= e^{\#}[x/e^{\#}(x) \cap v^{\#}] \\
\mathcal{R}^{\#}[\![this]\!]\, v^{\#}\, e^{\#}\, s^{\#} &= e^{\#}[this/e^{\#}(this) \cap v^{\#}] \\
\mathcal{R}^{\#}[\![i]\!]\, v^{\#}\, e^{\#}\, s^{\#} &= e^{\#} \\
\mathcal{R}^{\#}[\![null]\!]\, v^{\#}\, e^{\#}\, s^{\#} &= e^{\#} \\
\mathcal{R}^{\#}[\![x.i]\!]\, v^{\#}\, e^{\#}\, s^{\#} &= e^{\#}[x/e^{\#}(x) \setminus \{r^{\#} \mid r^{\#} \in dom(s^{\#}) \wedge \\
&\quad\ s^{\#}(r^{\#}) = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#} \rangle\rangle \wedge \\
&\quad\ n < i\}]
\end{aligned}
$$

1.2.3.2 Conditions

Le résultat d'une condition est donné par la fonction $\mathcal{B}$, qui renvoie deux environnements abstraits pour les deux résultats possibles de la condition, et le status indiquant la correction de ce point de programme.

$$\mathcal{B}^{\#} : \mathbb{C}\text{ond} \rightarrow \mathbb{E}nv^{\#} \rightarrow \mathbb{E}nv^{\#} \rightarrow \mathbb{S}tatus$$

La valeur abstraite et le statut des deux expressions sont évalués, ensuite l'environnement est restreint en fonction de l'opérateur conditionel, en enlevant les valeurs *noninit* dans l'environnement de sortie puisque si le code passe par la, ces valeurs ne seront forcément pas *noninit*. La correction du programme est vérifiée sans ces restrictions, de telle sorte que si une condition est évaluée avec un valeur contenant *noninit* le point de programme sera marqué comme incertain.

Note : l'évaluation du status n'est pas strictement déclarative, mais doit plutot etre évaluée dans l'ordre (d'abord vérifier s'il est incorrect, puis s'il est inconnu et enfin correct).

La comparaison d'égalité ci-dessous peut être faite entre deux objets (références) ou deux entiers,

on aura donc une série de conditions sur les ensembles abstraits pour s'assurer de comparer les bons types de valeur.

$$\mathcal{B}^{\#}[\![expr_1 = expr_2]\!] e^{\#} s^{\#} = \begin{cases} \langle e_t^{\#}, e_f^{\#}, status \rangle & si \quad \langle v_i^{\#}, status_i \rangle = \mathcal{E}^{\#}[\![expr_i]\!] e^{\#} s^{\#} \wedge \\ & \quad v^{\#} = v_1^{\#} \cap v_2^{\#} \setminus \{noninit\} \wedge \\ & \quad e_t^{\#} = \mathcal{R}^{\#}[\![expr_1]\!] v^{\#} (\mathcal{R}^{\#}[\![expr_2]\!] v^{\#} e^{\#} s^{\#}) s^{\#} \\ & \quad e_f^{\#} = \mathcal{R}^{\#}[\![expr_1]\!] v_{1*}^{\#} (\mathcal{R}^{\#}[\![expr_2]\!] v_{2*}^{\#} e^{\#} s^{\#}) s^{\#} \\ & \quad \text{avec } v_{1*}^{\#} = \begin{cases} v_1^{\#} \setminus v_2^{\#} & si \quad | v_2^{\#} | = 1 \\ v_1^{\#} & sinon \end{cases} \\ & \quad \text{et } v_{2*}^{\#} = \begin{cases} v_2^{\#} \setminus v_1^{\#} & si \quad | v_1^{\#} | = 1 \\ v_2^{\#} & sinon \end{cases} \\ & \quad status = incorrect \text{ si} \\ & \qquad status_1 = incorrect \vee \\ & \qquad status_2 = incorrect \vee \\ & \qquad v_1^{\#} = v_2^{\#} = \{noninit\} \vee \\ & \qquad v_{1,2}^{\#} \subseteq \{-, 0, +\} \wedge v_{2,1}^{\#} \subseteq \{null\} \cup \mathbb{R}ef^{\#} \vee \\ & \quad status = inconnu \text{ si} \\ & \qquad status_1 = inconnu \vee \\ & \qquad status_2 = inconnu \vee \\ & \qquad noninit \in v_1^{\#} \cup v_2^{\#} \vee \\ & \qquad a, b \in v_{1,2}^{\#} : a \in (\{null\} \cup \mathbb{R}ef^{\#}) \wedge b \in \{-, 0, +\} \\ & \quad status = correct \text{ sinon} \\ \langle \emptyset, \emptyset, incorrect \rangle & \text{sinon} \end{cases}$$

Même commentaire que ci-dessus, pour l'opérateur de comparaison d'ordre. Celui-ci n'est défini que pour les entiers entre-eux, il faut donc définir la relation d'ordre sur les ensembles abstraits.

$$\mathcal{B}^\#[\![expr_1 < \ expr_2]\!] e^\# s^\# \ = \ \begin{cases} \langle e_t^\#, e_f^\#, status\rangle & si \quad \langle v_i^\#, status_i\rangle = \mathcal{E}^\#[\![expr_i]\!] e^\# s^\# \wedge \\ & e_r^\# = \mathcal{R}^\#[\![expr_1]\!]\{-,0,+\}(\\ & \quad \mathcal{R}^\#[\![expr_2]\!]\{-,0,+\} e^\# s^\#) s^\# \\ & e_t^\# = \mathcal{R}^\#[\![expr_1]\!] v_{1_t}^\# (\mathcal{R}^\#[\![expr_2]\!] v_{2_t}^\# e_r^\# s^\#) s^\# \\ & e_f^\# = \mathcal{R}^\#[\![expr_1]\!] v_{1_f}^\# (\mathcal{R}^\#[\![expr_2]\!] v_{2_f}^\# e_r^\# s^\#) s^\# \\ & \text{avec} \\ & v_{1_t}^\# = \begin{cases} \{-\} & si \quad + \notin v_2^\# \\ \{-,0,+\} & si \quad + \in v_2^\# \end{cases} \\ & v_{1_f}^\# = \begin{cases} \{-,0,+\} & si \quad - \in v_2^\# \\ \{0,+\} & si \quad - \notin v_2^\# \wedge 0 \in v_2^\# \\ \{+\} & sinon \end{cases} \\ & v_{2_t}^\# = \begin{cases} \{+\} & si \quad - \notin v_1^\# \\ \{-,0,+\} & si \quad - \in v_1^\# \end{cases} \\ & v_{2_f}^\# = \begin{cases} \{-,0,+\} & si \quad + \in v_1^\# \\ \{-,0\} & si \quad + \notin v_1^\# \wedge 0 \in v_1^\# \\ \{-\} & sinon \end{cases} \\ & status = incorrect \text{ si} \\ & \quad status_1 = incorrect \vee \\ & \quad status_2 = incorrect \vee \\ & \quad v_1^\# \cap \{-,0,+\} = \emptyset \vee \\ & \quad v_2^\# \cap \{-,0,+\} = \emptyset \\ & status = inconnu \text{ si} \\ & \quad status_1 = inconnu \vee \\ & \quad status_2 = inconnu \vee \\ & \quad (v_1^\# \cup v_2^\#) \not\subseteq \{-,0,+\} \\ & status = correct \text{ sinon} \\ \langle \emptyset, \emptyset, \emptyset, \emptyset, incorrect\rangle & \text{sinon} \end{cases}$$

1.2.4 Assignations

Les asignations vont modifier le store abstrait puisqu'il faudra modifier la valeur d'une variable ou d'un champ d'objet.

$$\mathcal{A}^\# : \mathbb{A}\text{ss} \to \mathbb{E}nv^\# \to \mathbb{S}tore^\# \to (\mathbb{E}nv^\# \times \mathbb{S}tore^\# \times \mathbb{S}tatus)$$

Dans le cas d'une simple assignation, on évalue l'expression et on replace la valeur abstraite courante de la variable dans l'environnement abstrait, par la nouvelle retournée par l'évaluation de l'expression. Le status retourné est celui de l'évaluation de l'epxression puisque l'assignation ne peut pas être une erreur.

Dans le cas d'une assignation à un champ d'objet, on va changer dans le store l'ancienne instance abstraite par une nouvelle dont la valeur abstraite du champ demandé a été mise à jour avec l'évaluation de l'expression donnée. Une erreur apparait si le champ demandé n'existe pas dans l'ensemble des instances abstraites représentées par la variable x.

Enfin la création d'un nouvel objet demande la création d'une nouvelle instance abstraite de la classe demandée, avec toutes les valeurs abstraites mises à *noninit*.

$$\mathcal{A}^{\#}[\![x = expr]\!]\, e^{\#}\, s^{\#} = \langle e^{\#}[x/v^{\#}], s^{\#}, status\rangle : v^{\#}, status = \mathcal{E}[\![expr]\!]\, e^{\#}\, s^{\#}$$

$$\mathcal{A}^{\#}[\![x.i = expr]\!]\, e^{\#}\, s^{\#} = \begin{cases} \langle e^{\#}, s^{\#}[r^{\#}/ins^{\#}], status\rangle & \text{si } \mathcal{D}^{\#}[\![x.i]\!]\, e^{\#}\, s^{\#} =< r^{\#}, i > \wedge \\ & s^{\#}(r^{\#}) = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#}\rangle\rangle \wedge \\ & ins^{\#} = \langle n, \langle v_1^{\#}, \ldots, v_{i'}^{\#}, \ldots, v_n^{\#}\rangle\rangle \wedge \\ & v_{i'}^{\#}, status = \mathcal{E}[\![expr]\!]\, e^{\#}\, s^{\#} \\ \langle \emptyset, \emptyset, incorrect\rangle & \text{sinon} \end{cases}$$

$$\begin{aligned} \mathcal{A}^{\#}[\![x = \text{new}/i]\!]\, e^{\#}\, s^{\#} = {} & \langle e^{\#}[x/r^{\#}], s_*^{\#}, correct\rangle : \\ & s_*^{\#}(r^{\#}) = \langle i, \langle v_1^{\#}, \ldots, v_i^{\#}\rangle\rangle \\ & v_1^{\#}, \ldots, v_i^{\#} = \{noninit\} \wedge \\ & r^{\#} \in \mathbb{R}ef^{\#} \wedge dom(s_*^{\#}) = dom(s^{\#}) \cup \{r^{\#}\} \end{aligned}$$

1.3 Relations de transitions abstraites

Nous decrivons mainteant comment faire évoluer l'état du programme abstrait en fonction des instructions rencontrée, en utilisant les fonctions abstraites définies plus haut.

1.3.1 Assignation

Seul l'environnement et le store sont mis à jour après une assignation.

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l\ ass\ l_{out}}{\longrightarrow} < l_{out}, e^{\#\prime}, s^{\#\prime}, P^{\#} >, status$$

On utilise simplement la fonction sémantique abstraite $\mathcal{A}^{\#}$ pour les obtenir :

$$< e^{\#\prime}, s^{\#\prime} >= \mathcal{A}^{\#}[\![ass]\!]\, e^{\#}\, s^{\#}$$

1.3.2 Branchement conditionnel

Le branchement produit deux nouveaux environnements selon la condition et saute au label associé.

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l \text{ if } cond \text{ then } l_1 \text{ else } l_2}{\longrightarrow} < l_1, e_t^{\#}, s^{\#}, P^{\#} >, < l_2, e_f^{\#}, s^{\#}, P^{\#} >, status$$

De nouveau, on utilise simplement la fonction abstraite $\mathcal{B}^{\#}$

$$< e_t^{\#}, e_f^{\#}, status >= \mathcal{B}^{\#}[\![cond]\!]\, e^{\#}\, s^{\#}$$

1.3.3 Entrées/Sorties

Les entrées/sorties ont été complètement abstraites dans l'état du programme pour faciliter les choses. On donc assigner aux valeurs de celles-ci simplement tous les entiers possibles, ou dans le cas d'une sortie, vérifier que le nombre imprimé est bien un entier potentiellement et pas une référence.

1.3.3.1 Input

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l \in read\, x\, l_{out}}{\longrightarrow} < l_{out}, e^{\#'}, s^{\#'}, P^{\#} >, status$$

avec

$$< e^{\#'}, s^{\#'}, status >= \mathcal{A}[\![x = i]\!]\, e^{\#}, s^{\#}$$

1.3.3.2 Output

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l\, \text{write}\, x\, l_{out}}{\longrightarrow} < l_{out}, e^{\#}, s^{\#}, P^{\#} >, status$$

avec

$$status = \begin{cases} correct & \text{si } e^{\#}(x) \subseteq \{-, 0, +\} \\ incorrect & \text{si } e^{\#}(x) \cap \{-, 0, +\} = \emptyset \\ inconnu & \text{sinon} \end{cases}$$

1.3.4 Appel de méthode abstrait

Ceci est peut-être la partie la plus complexe à prendre en compte pour abstraire correctement l'exécution du programme, en effet la pile abstraite est utilisée lors des appels, et il faut bien sur que celle ci ne grandisse pas indéfiniment, pour que l'algorithme puis détecter les états déjà explorés.

1.3.4.1 Table des méthodes

On supposera qu'il existe une table reprenant toutes les méthodes définies de la forme :

$$m/i\,(z_1, \ldots, z_n)\, l_0\, \{instructions\}\, l_f\, x_f$$
$$\vdots$$

qui contiendra pour chaque méthode :
- m/i ou m, le nom de la méthode et son niveau si elle est dynamique.
- $(z_1, \ldots, z_n)$, la liste des paramètres formels.
- l_0, le label de début de méthode.
- l_f, le label de fin de méthode.
- x_f, la variable dans laquelle prendre le résultat de la méthode.

On ajoutera l'indice i aux éléments ci-dessus lorsque plusieurs méthodes sont considérées, indiquant le niveau de la méthode m.

1.3.4.2 Appel de méthode statique

L'appel statique est le plus simple des trois types d'appels. Il suffit de s'assurer que la méthode statique existe, puis mettre dans la pile un état possible (l'état courant) d'appel de cette méthode, avec le label de retour.

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l\, x_1 = m(y_1, \ldots, y_n)\, l'}{\longrightarrow} < l_0, e'^{\#}, s^{\#}, P'^{\#} >, status$$

Nous définissons d'abord l'environnement $e^{\#}_*$ qui contient toutes les variables de l'environnement courant en enlevant, si besoin est et pour les paramètres $y_1, \ldots, y_n$, la valeur *noninit* de l'ensemble des valeurs abstraites, puisque si la méthode est effectivement appelée, tous les paramètres sont initialisés.

Il faut ensuite définir quel sera l'environnement dans lequel s'exécutera la méthode, $e'^{\#}$. Celui-ci contiendra la version locale des paramètres ($z_1, \ldots, z_n$ donnés dans la définition de la méthode) ainsi qu'une définition pour *this*.

Enfin la pile contiendra pour le label de retour de méthode l_f les tuples précédents pour le label ainsi qu'un nouveau tuple contenant le label de retour dans le programme l', l'environnement courant à restaurer lors du retour, et la variable dans laquelle mettre le résultat de l'appel.

Le status est défini en fonction de la présence des paramètres et de leur valeur différente de *noninit*

$$
\begin{aligned}
\forall x \in dom(e^{\#}), e^{\#}_*(x) = & \begin{cases} e^{\#}(x) \setminus \{noninit\} & \text{si} \quad x \in \{y_1, \ldots, y_n\} \\ e^{\#}(x) & \text{si} \quad x \notin \{y_1, \ldots, y_n\} \end{cases} \\
dom(e'^{\#}) = & \{z_1, \ldots, z_n, this\} \\
\forall z \in dom(e^{\#}_*), e'^{\#}(z) = & e^{\#}_*(z) \text{ et } e'^{\#}(this) = \{noninit\} \\
P'^{\#} = & P^{\#}\left[l_f / P^{\#}(l_f) \cup \{\langle l', e^{\#}_*, x_f \rangle\}\right] \\
status = & \begin{cases} correct & \text{si} \quad \forall i : 0 \leq i \leq n, e^{\#}(y_i) \neq \{noninit\} \\ incorrect & \text{si} \quad \exists m \in TableMethodes \vee \exists i : 0 \leq i \leq n, e^{\#}(y_i) = \{noninit\} \\ inconnu & \text{sinon} \end{cases}
\end{aligned}
$$

1.3.4.3 Appel de méthode dynamique

Pour l'appel dynamique, il va falloir de nouveau interpréter plusieurs codes en parallèle puisque l'objet sur lequel la méthode est appellée n'est peut être pas défnini de manière univoque (la variable x_2 est un valeur abstraite, et contient donc un ensemble de valeurs possibles pour les références, il se pourrait donc que x_2 soit un objet de classe 2 ou 3 ou 5, il faudra donc effectuer la recherche des méthode dans la table pour chaque niveau possible de l'objet.

Nous calculerons dès lors l'ensemble E qui contiendra chaque fois un label de début de méthode, un environnement pour cette méthode et la pile abstraite.

$$
< l, e^{\#}, s^{\#}, P^{\#} > \overset{l\, x_1 = x_2.m(y_1,\ldots,y_n)\, l'}{\longrightarrow} E = < l'_i, e'^{\#}_i, s^{\#}, P'^{\#}_i >^*, status
$$

Nous calculons l'ensemble $refs^{\#}$ des références abstraites possible que contient la variable x_2, en s'assurant qu'il existe une méthode pour chaque instance pointée par la référence abstraite grâce à la function $levels(m)$ qui renvoie l'ensemble des niveaux d'objets pour lesquels une méthode m existe.

Ensuite vient la définition du contenu de E en partant des références calculées ci-dessus. Pour chaque instance le niveau (i) de la méthode qu'il faut appeler étant donné le niveau (n) de l'objet est donné par la fonction $level(m,n)$. Celle-ci renvoie le numéro de méthode m le plus proche du numéro n (inférieur ou égal) pour laquelle une méthode m est définie.

Lorsque le numéro i de la méthode est connu, l'élement à ajouter dans E contiendra le label l_i de début de la méthode m de niveau i, un nouvel environnement pour la méthode, qui contiendra la valeur abstraite des paramètres passés à la méthode (en enlevant *noninit* comme précédament) et la variable *this* qui prendra la valeur de la référence abstraite $r^{\#}$ contenu dans la valeur abstraite de x_2, et la nouvelle pile définie après.

La nouvelle pile contient un nouvel état, dans lequel est enregistré le tuple contenant le label de retour dans le code appellant, l'environnement courant et la variable x_1 qui contiendra la valeur abstraite résultat de la méthode appellée.

Le status est défini en fonction de l'existence d'une ou plusieurs méthodes dynamiques pour les valeurs abstraites que peut avoir avoir la variable x_2. Il faut aussi s'assurer que les paramètres passés soient effectivement initialisés.

$$
\begin{array}{l}
refs^{\#} = \{r^{\#} \mid \forall\, r^{\#} \in e^{\#}(x_2) : r^{\#} \in \mathbb{R}ef^{\#} \wedge s^{\#}(r^{\#}) = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#} \rangle \rangle \wedge n \in levels(m)\} \\
\forall\, r^{\#} \in refs^{\#}, \exists e \in E, r^{\#} = \langle n, \langle v_1^{\#}, \ldots, v_n^{\#} \rangle \rangle \wedge i = level(m,n) \wedge \\
e =< l_i, e_i^{'\#}, s^{\#}, P_i^{\#} > \wedge \forall j : 1 \leq j \leq n, e_i^{'\#}(z_j) = e^{\#}(y_j) \setminus \{noninit\} \wedge \{r^{\#}\} = e_i^{'\#}(this) \wedge \\
P_i^{\#}(l_{fi}) = P^{\#}(l_{fi}) \cup \langle l', e^{\#}, x_1 \rangle \\
status = \left\{ \begin{array}{lll}
correct & \text{si} & \forall j : 0 \leq j \leq i, e^{\#}(y_j) \neq \{noninit\} \wedge \\
 & & refs^{\#} \neq \emptyset \\
incorrect & \text{si} & \exists j : 0 \leq j \leq i, e^{\#}(y_j) = \{noninit\} \vee \\
 & & refs^{\#} = \emptyset \\
inconnu & \text{sinon} &
\end{array} \right.
\end{array}
$$

1.3.4.4 Appel de la méthode *super*

L'appel de méthode *super* est similaire à l'appel dynamique, si l'on considère que l'on fait en réalité $this.m(y_1, \ldots, y_n)$, en restreignant la recherche de méthode dans la table comme défini au dessus, au niveau maximal = *courant* − 1 pour chaque valeur concrète possible que prendra *this*.

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{l\, x_1 = super.m(y_1,\ldots,y_n)\, l'}{\longrightarrow} E =< l'_i, e'^{\#}, s^{\#}, P'^{\#} >^*, status$$

1.3.4.5 Retour de méthode

Lors d'un retour de méthode, l'interprétation abstraite doit retrouver tous les labels de retour de méthode associés à la méthode qui se termine. Il faut ensuite exécuter virtuellement en parallèle l'interprétation de tous ces points de retour avec la valeur de la variable de retour et en restaurant l'environnement stocké sur la pile.

Tous ces points de retour seront donnés dans l'ensemble E

$$< l, e^{\#}, s^{\#}, P^{\#} > \overset{m/i(z_1,\ldots,z_n)\, l_0\, \{instructions\}\, l_f\, x_f}{\longrightarrow} E = (< l'_i, e^{\#'}, s^{\#}, P^{\#'} >)^*$$

La définition du contenu de E, qui exprime le fait que l'environnement de retour est ajusté pour contenir la valeur abstraite de la variable de retour calculée par la méthode, en enlevant la valeur *noninit* de cet ensemble, puisqu'une méthode ne pourra jamais renvoyer une variable noninit. Le status est défini en tenant compte de la possibilité d'un retour d'une variable *noninit* auquel cas le status sera inconnu ou incorrect.

La pile abstraite n'est pas modifiée, puisqu'il faut retenir quels sont les états que l'on a visité.

$$\forall \langle l', e^{\#'}, x \rangle \in P^{\#}(l_f), \exists e,$$
$$e = \langle e^{\#'}[x/e^{\#}(x_f) \setminus \{noninit\}], s^{\#}, P^{\#} \rangle$$
$$status = \begin{cases} correct & \text{si} \quad noninit \notin e^{\#}(x_f) \\ incorrect & \text{si} \quad e^{\#}(x_f) = \{noninit\} \\ inconnu & \text{sinon} \end{cases}$$

1.3.5 Lancement d'un programme

Lors du lancement de l'interprétation abstraite d'un programme, la méthode statique sans arguments nommée "Main" est appelée, avec un environnement abstrait vide, un store abstrait vide et une pile contenant le label final du programme.

Implémentation

Nous avons réalisé l'implémentation de l'analyseur abstrait en utilisant l'algorithme univariant. L'algorithme repose sur une itération autour d'un point fixe. Il faut donc qu'il y ait une évolution d'un état pour déterminer l'avancement de l'algorithme. Pour déterminer si l'état abstrait évolue, il faut introduire un définition d'ordre sur l'état abstrait. Plus précisément nous avons besoin de la relation $\leq$ entre deux états $\alpha_1^\#$ et $\alpha_2^\#$.

$$\forall \alpha_1^\# \in \mathbb{E}tat^\#, \forall \alpha_2^\# \in \mathbb{E}tat^\# : \alpha_1^\# \leq \alpha_2^\# \text{ ssi}$$
$$\alpha_1^\# = \langle l, e_1^\#, s_1^\#, p_1^\# \rangle \wedge \alpha_2^\# = \langle l, e_2^\#, s_2^\#, p_2^\# \rangle \wedge e_1^\# \leq e_2^\# \wedge s_1^\# \leq s_2^\# \wedge p_1^\# \leq p_2^\#$$

Il reste encore à définir la relation $\leq$ entre les environnements, les stores et les piles abstraits.

$$\forall e_1^\# \in \mathbb{E}nv^\#, \forall e_2^\# \in \mathbb{E}nv^\# : e_1^\# \leq e_2^\# \text{ ssi}$$
$$dom(e_1^\#) \subseteq dom(e_2^\#) \wedge \forall x \in dom(e_1^\#) : e_1^\#(x) \leq e_2^\#(x)$$

$$\forall s_1^\# \in \mathbb{S}tore^\#, \forall s_2^\# \in \mathbb{S}tore^\# : s_1^\# \leq s_2^\# \text{ ssi}$$
$$dom(s_1^\#) \subseteq dom(s_2^\#) \wedge \forall r^\# \in dom(s_1^\#) :$$
$$s_1^\#(r^\#) = \langle n, \langle v_{1,1}^\#, \ldots, v_{1,n}^\# \rangle \rangle \wedge$$
$$s_2^\#(r^\#) = \langle n, \langle v_{2,1}^\#, \ldots, v_{2,n}^\# \rangle \rangle \wedge$$
$$\forall i : 0 \leq i \leq n, v_{1,i} \subseteq v_{2,i}$$

$$\forall p_1^\# \in \mathbb{P}ile^\#, \forall p_2^\# \in \mathbb{P}ile^\# : p_1^\# \leq p_2^\# \text{ ssi}$$
$$dom(p_1^\#) \subseteq dom(p_2^\#) \wedge \forall l_1 \in dom(p_1^\#) :$$
$$\forall \langle l_2 \times x \times e_1^\# \rangle \in p_1^\#(l_1), \exists \langle l_2, x, e_2^\# \rangle \in p_2^\#(l_1) : e_1^\# \leq e_2^\#$$

2.1 Structures de Données

Nous avons suivit scrupuleusement les définissions que nous donnons dans la section précédente. Ce qui nous a mené à pas mal de problèmes car certaines de des définissions étaient erronées.

Les principales classes sont *AbstractState, AbstractEnvironnement, AbstractStack, AbstractStore* et *AbstractValues*. Une attention particulière est portée sur la classe *AbstractValues* pour la rendre la plus efficace possible dans les calculs d'intersection, d'union et d'ordre. Nous avons utilisé des valeurs booléennes pour représenter les différents valeurs abstraite possible. Et des tables de hashage pour les représentation d'ensembles.

2.2 Interprétation abstraite

En premier lieu nous avons ajouté un interpréteur concret à notre compilateur. Cela nous à permis d'avoir une première vue sur ce qu'est un interpréteur. Ensuite nous avons généralisé l'interprétation

pour pouvoir la rendre soit concrète soit abstraite.

Ici aussi nous avons suivit scrupuleusement les définissions de la section précédente, ce qui nous a à nouveau posé pas mal d'ennuis car certaines de ces définissions étaient erronées. Nous avons donc passé beaucoup de temps à trouver les erreurs de définissions et à corriger les parties de code correspondante.

Au final nous sommes arrivé a interpréter les code de manière abstraite avec des résultats satisfaisant.