Chapter 7

Question 15: Expliquez ce qui a été changé par rapport au langage du chapitre 6 pour prendre en compte l ´évaluation paresseuse.

Au langage du chapitre 6 quand on fait une substitution on évalue l'expression tandis que avec l'evaluation paresseuse on fait une affectation sur l'expression mais on ne l'évalue pas qu'en cas où une décision doit être prise comme c'est le cas avec les structures de controle (if then else) ou (while) . Dans ce cas on doit décider quel block de program executer donc on fait une évaluation imedaite de notre expression. L'affectation (qu'on note avec :=) est une règle paresseuse car quand on affect une variable à une expression on est pas obligé de calculer cet expression. L'appel d'une fonction est aussi une règle paresseuse car on dèfinie une fonction f(x,y) et puis on l'appelle en lui passant des differentes variables comme par exemple f(e1,e2).

Pour differencier au langage du chapitre 6 où on avait une assosiation entre les variables et les valeurs, dans une evalutation paresseuse on a plutot une association entre des variables et des expressions. Donc on doit pouvoir garder des expressions qui sont composées des variables non évaluée dans la mémoire ce qui rend l'evaluation paresseuse baucoup plus abstraite qu'une évaluation immédiate.

Ca c'est donc le principe d'évaluation paresseuse. Maintenant on doit décider quelles règles du chapitre 6 on peut les transformer pour les mettre dans la catégorie paresseuse ou immédiate.

On commence par la règle d'affectation: au chapitre 6 on avait ca:

Definition (Sémantique d'évaluation : Règle affectation)

$e \in ExprVar_V$ et $v \in V$, $S, S', S'' \in Subs$

$$Raffectation : \frac{S \vdash e \Longrightarrow n}{S \vdash v := e \Longrightarrow_I S/[v = n]}$$

Donc, étant donnée une substitution S, on a l'affectation v prend la valeur de e ($S \vdash v := e$) ce $S \vdash e \Longrightarrow n$ qui implique qu'on doit d'abord évaluer e , et depuis S, e s'évalue en n et la valeur de v prend la valeur n.

Pour l'évaluation paresseuse de la règle d'affectation on doit tenir compte de la notion de sequence, c'est à dire que quelque part la substitution change et le meme nom de variable va prendre des valeurs differentes. Donc on doit etre capable de dire qu'une expression on la transforme à une autre expression par le biais d'une substitution.

Definition (Sémantique d'évaluation : Règle affectation)

$e \in ExprVar_V$ et $v \in V$, $S \in Subs$

$$Rlaffectation : \frac{e \Longrightarrow_S e'}{S \vdash v := e \Longrightarrow_I S/[v = e']}$$

Ici, il est important qu'on ait plus des variables dans e' qui ne sont pas définis ou qui sont défnis dans la substitution sinon on pourra pas tenir compte de l'état correct des variables.

Donc on évalue e en e' et depuis la substitution S , v prend la valeur de e'. Mais ca c'est vrai étant donnée que $v \in Dom(S)$ (v est bien défini dans la substitution S). C'est pour cette raison qu'on défini les règles suivantes:

$\frac{v \notin Dom(S)}{v \overset{\Rightarrow}{S} v}$ Donc on ne change pas la variable car c'est possible que la variable n'est pas défini.

$\frac{v \in Dom(S)}{v \overset{\Rightarrow}{S} [v = e] e}$ Là on a que v $\in Dom(S)$ donc e est le resultat de l'expression.

Pour les valeurs (constants): $\frac{\square}{n \overset{\Rightarrow}{S} n}$

Pour les expressions: $\frac{e \overset{\Rightarrow}{S} e'', e' \overset{\Rightarrow}{S} e'''}{e + e' \overset{\Rightarrow}{S} e'' + e'''}$

Les fonctions: $\frac{e_1 \overset{\Rightarrow}{S} e_1', \ldots, e_n \overset{\Rightarrow}{S} e_n'}{f(e_1, \ldots, e_n) \overset{\Rightarrow}{S} f(e_1', \ldots, e_n')}$

Les règles *if then else* et *while* restent identique comme au langage du chapitre 6 car on est obligé de faire une évaluation immédiate. Pour les rélation (<, >, >= , <=) les règles restent les memes aussi car l'évaluation est "eager" donc il y a une décision à faire.

Au niveau de l'évaluation d'un programme, ca va pas faire un changement massive. Il faut juste s'assurer qu'on a fait une évaluation complet (final) et qu'il n'y a plus des instructions à executer une fois le programme terminé.

La règle d'évaluation:

Definition (Sémantique d'évaluation d'un programme)

$p \in Bloc_V$, $S, S', S'' \in Subs$

$$\frac{S \vdash p \Longrightarrow_B S'', S'' \Longrightarrow S'}{S \vdash p \Longrightarrow_P S'}$$

Si la s $RSubst : \frac{S \Longrightarrow S', S \vdash e \Longrightarrow n}{S/[v = e] \Longrightarrow S'/[v = n]}$ t vide : $\frac{}{\epsilon \Longrightarrow \epsilon}$ donc on évalue une substitution vide.

Sinon: donc S' s'évalue avec la nouvelle valeur prise par la variable v. Donc ici on termine et on fait l'évaluation concret de notre expression.

Question 16: Expliquez toutes les étapes nécessaires à la création d'une sémantique pour un langage de programmation

Pour donner la définition d'une semantique pour un langage de programmation il n'y a pas une "recette" facile à suivre et la procédure et plutot abstraite.

1. On commence tout d'abord par définir une syntaxe pour notre langage.
2. En suite il faut pouvoir définir ce qu'on apelle une domaine d'intérpretation sémantique et
3. Il faut définir un contexte pour chaque categorie syntaxique.
4. Ensuite on doit définir les rélations d'évaluation sémantique pour chaque catégorie syntaxique et
5. Finalement on doit construire les régles d'inference des rélations d'évaluation.

1. Pour une syntaxe (abstraite) on rencentre souvent les notions d'une variable(X), une fonction(F), etc. Donc on doit définir les noms des domaines syntaxique qui sont des ensembles dans lequels on a des objets syntaxiques qui sont bien définis comme par exemple: $F, X, \exp r_{F,X}$, etc. Pour définir une domaine syntaxique il y a des differentes méthodes qu'on a vu avant, donc soit inductivement ou avec une domaine prédéfini des termes $T_{OP(D)}$.

2. Un autre pas important pour la définition d'une semantique est de définir un domaine d'interpretation semantique qui est souvent un choix qui n'est pas si immédiat. Par exemple quand on fait la sémantique "eager" on a le domaine semantique qui est des substitutions qui associent des valeurs aux variables. Dès qu'on est a changé la semantique "eager" à "lazy" on a du changer aussi le domaine d'interpretation semantique car au lieu d'associer des valeurs aux variables on associe des expressions aux variables. Par exemple pour une semantique eager les expressions sont souvent des entier naturels (v = n). Pour les expressions dans une machine abstraite le domaine d'interpretation serait la pile. Pour intepreter il est important de définir donc la substitution des variables car il y a des moments dans notre programme où on doit faire un choix ou décision et on aimerait pouvoir interpreter ces variables.

3. On sait qu'on va avoir des differents categories syntaxiques (les blocs, les instructions, les expressions) et on aimerait savoir dans quel contexte on doit évaluer ces differentes constructions. Par exemples, pour les expressions souvent le context c'est les substitutions. Mais pour un bloc on doit pas seulement connaitre les expressions mais aussi le nom de la fonction, donc si on évalue un bloc il nous faut un contexte qui est lié aux fonctions et aux substitutions.
4. Une fois qu'on a les notions de contexte d'évalution on peut construire une rélation $(C \vdash a \overset{\Rightarrow}{} lr ¿$. Donc en fonction d'un contexte "C" on contruit une rélation (de "a" on produit un résulat "lr") avec "a" comme une catégorie syntaxique (epression, instruction ou bloc).

5. La dernière étape et donc de construire les règles d'inférence pour décrire comment on construit cette rélation.

Chapitre 8

Question 17: Expliquez le processus pour interpréter un programme

Side note: vous pouvez utiliser la question 16 pour pouvoir décrire la construction de semantique

L'idée est de créer un programme qui prend en entrée le langage qu'on veut interpreter et qui produit comme resultat l'état du programme après l'exécution. Prenons comme exemple le langage Prolog, l'interpreatation c'est fait en suivant qulques étapes:

1. On construit la semantique du langage de programme (Prolog)
2. On transforme cette semantique sous forme Prolog
    a. interpreter la syntaxe
    b. la semantique
3. Apres on peut:
    a. soit directement evaluer des expressions
    b. soit faire une analyse de domaine (execution symbolique) c.à.d au lieu de regarder tous les calclus pour des valeurs specifiques on peut le faire sur des domaines de calculs

Il y a quelques principes à appliquer pour faire la transformation de la semantique sous forme Prolog:

- Premierement il faut établier la structure syntaxique du langage en prolog. On peut soit essayer de construire des termes prolog soit on fait un parseur.
- On doit ensuite définir la notion d'état du programme c.à.d la domaine sémantique.
- Définir des procédures de manipulation de notre langage (substitutions, listes, etc)
- Finalelement il faut construire les rélations. Traduire en prolog les règles sémantiques sans en changer le sens tout en tenant compte de:
    - filtrer les règles applicables de manière unique (determinisme)
    - evaluer la péconditions
    - calculer l'effet de la regle

Il faut donc faire attention de ne pas avoir de non-déterminisme, c.a.d on a pas filtrer les règles d'une maniere unique. Eventuellement on peut avoir des problèmes de négation.

Question 18: Expliquez l'intérêt et la démarche pour l'analyse de programmes.

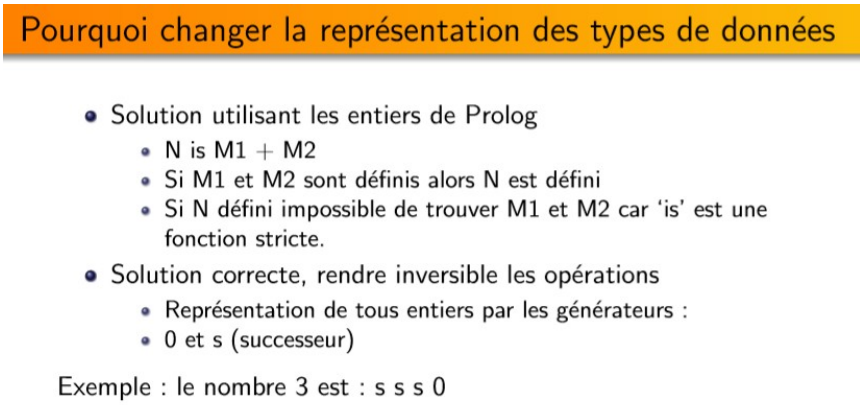
Une fois qu'on a défini la sémantique de notre langage, on peut l'interpreter pour analyser notre programme et étudier son comportement pour trouver les chemins d'execution possibles ou de détecter des anomalies telle que des variables non initialisées, code mort etc.

La démarche qu'on doit suivre:

- interpreter notre langage un utilisant l'interprète généré à partir de la semantique
  - exécution directe
  - ecécution symbolique:
    - posséder une axomatisation des types de données
    - avoir un mécanisme d'évaluation plus "intelligent" qu la résolution SLD
- Analyse statique

Side note: il n y avait vraiment pas plus d'infos sur cette question. Si je toruve une réponse plus avancée je vous laisse savoir, sinon je vous encourage de faire plus attention sur cette question car la réponse est assez courte

Question 19: Expliquez comment et pourquoi il peut être intéressant de changer la représentation des types de données.



Il y a que ce slide qui parle sur le changement de la représentation des types de données. Pris dans le chapitre 8 page 28.