UFR 919 Sciences de l'ingénieur
Master Informatique – Spécialité STL
Analyse de Programmes et Sémantique (APS)

TD9 — Sémantique Dénotationnelle
Extensions à BOPL

Jacques Malenfant, Olena Rogovchenko

1 Forever `for`

On s'attaque encore une fois à la boucle `for`. En vous inspirant fortement du travail fait pour définir la sémantique dénotationnelle de la boucle `for` dans le mini-langage au TD6, rajoutez le traitement de la boucle **for** dans la sémantique dénotationnelle de *BOPL*.

2 Constructeurs

On veut maintenant étendre la syntaxe du `new`, pour permettre l'initialisation des champs d'un objet lors de sa création. Apportez les modifications nécessaires pour permettre l'utilisation des constructeurs avec la syntaxe concrète de la forme :

`new` <nom_classe> `(`<liste_arguments>`)`

le constructeur lui-même étant défini comme une méthode de la classe dont le nom est celui de la classe.

Quelle solution donneriez-vous si on n'a qu'un seul constructeur dans la classe d'instantiation ?

Comment prendriez-vous en compte le fait qu'il puisse y avoir surcharge des constructeurs (plusieurs constructeurs avec un nombre ou des types d'arguments différents, comme en Java) ? Comment traiteriez-vous le chaînage des constructeurs de la classe d'instantiation vers ses super-classes ?

3 Lecture sur un flux d'entrée

BOPL possède une instruction `writeln` e qui permet d'écrire la valeur d'une expression sur la sortie standard. On propose maintenant d'ajouter l'opération symétrique `readln` v qui permettra de lire une valeur sur l'entrée standard et de la ranger dans la variable donnée v.

Un peu comme la sortie standard, l'entrée standard doit être représentée par un paramètre de la fonction de valuation du programme. Et, pour simplifier, elle le sera par une liste de chaînes de caractères, chacune représentant un nombre ou l'une des chaînes de caractères `true` et `false` représentant les booléens.

Donnez les extensions de la grammaire, des domaines sémantiques et des fonctions de valuation nécessaires pour arriver à définir la sémantique de l'instruction `readln` *id*
Vous pouvez supposer qu'il existe les fonctions suivantes :

— `string->Z` : **String**→**Z** pour la conversion des chaînes en entiers relatifs,

— `string->T` : **String**→**T** pour la conversion des chaînes en booléens,
— *is***Integer** : **String**→**T** qui retourne vrai si la chaîne représente un entier, et
— *is***Boolean** : **String**→**T** qui retourne vrai si la chaîne représente un booléen.

Ajoutez maintenant la possibilité de ranger une valeur directement dans le champ d'un objet, c'est-à-dire qu'au lieu d'avoir une variable, on puisse avoir le champ d'une objet qui soit mentionné dans l'instruction : `readln` *e id*.

4 Partie TME

4.1 Implantation du `for` et des constructeurs

Reprenez les solutions des exercices du TD et introduisez-les dans l'implantaiton en Scheme de la sémantique dénotationnelle de BOPL.

4.2 Traitement du `return`

Dans l'implantation actuelle de BOPL, un `return` n'engendre pas de sortie du bloc de contrôle, mais uniquement le rangement de la valeur de retour dans une variable réservée à cet effet. Quels sont les problemes posés par une telle implantation ?

Quelles règles d'exécution du programme faut-il modifier pour obtenir une sortie du bloc de contrôle après un `return` ? Expliquez votre approche.