Sémantique et TDL

Projet

Compilation du langage Micro-Java

1 But du projet

Il s'agit ici d'écrire un compilateur pour le langage Micro-Java dont la grammaire est donnée en annexe. Ce compilateur devra engendrer du code pour la machine virtuelle TAM et sera conçu dans l'idée d'engendrer du code pour les assembleurs x86 et Sparc avec le minimum de changement.

Parmi les concepts présentés par micro-java, on peut citer

1. La notion d'importation de classes
2. La définition d'une classe et du type associé
3. L'héritage et le sous-typage associé
4. Quelques opérations arithmétiques et booléennes
5. L'accès aux attributs d'une instance
6. L'appel de méthodes par liaison tardive
7. La visibilité (publique, privée) des attributs et méthodes
8. L'utilisation du this dans les expressions du langage et dans les constructeurs
9. L'utilisation du super

NB. Les six premiers font partie du minimum à réaliser.

2 Moyens et conseils

Le compilateur sera écrit en utilisant Java et le générateur de compilateur EGG déjà étudié. L'utilisation d'Eclipse permet de gagner du temps, mais n'est pas obligatoire.

Le projet sera bien sûr basé sur

- Une gestion de la table des symboles permettant de conserver des informations sur les classes (héritage, types et sous-types, ...), attributs (visibilité, type, ...), méthodes (signature, ...), variables locales (type, adresse, ...), etc. La gestion de cette TDS devra être votre premier travail car tout dépend d'elle que ce soit pour le typage ou la génération de code. Donc, pas de précipitation, il sera difficile de revenir en arrière si vos choix s'avèrent peu pertinents.
- Le contrôle des types. On réfléchira particulièrement au traitement du sous-typage et son rapport avec l'héritage.
- La génération de code. TAM est le plus simple des assembleurs pour la génération, mais essayez d'être le plus générique possible pour éventuellement traiter les autres cibles. L'appel de méthode par liaison tardive est LA difficulté du projet.

Vous avez toute liberté pour l'organisation du travail dans le groupe, mais n'oubliez pas que pour atteindre votre objectif dans les délais, vous devez travailler en étroite collaboration, surtout au début pour la conception de la TDS. N'hésitez pas à nous poser des questions, (mail, en TD, en TP) si vous avez des doutes sur votre conception.

3 Dates, Remise

Le projet a commencé ...

Votre trinome doit être constitué avant le mardi 13 mai 2008 et ne devra comporter que des membres de votre groupe de TD. Vous en informerez votre enseignant de TD.

Le projet se terminera le jeudi 12 juin 2008. Les tests auront lieu le même jour de 8h à 12h.

Les sources (projet Eclipse et version 'make'), la documentation (TAM, X86 et SPARC) et le présent sujet sont dans /usr/local/gen6/0708.

Un document imprimé ou manuscrit (si votre écriture le permet) expliquant vos choix et limitations (ou extensions) dans le traitement de micro-java sera remis au moment du test. Ce document ne sera pas long, mais le plus précis possible (schémas) pour nous permettre de comprendre et juger votre travail.

Les fichiers de votre projet (export Eclipse, tar ou zip) seront envoyés par mail avant le test à votre enseignant de TD.

Bon courage à tous.

4 Grammaires

La grammaire de Micro-Java est donnée sous deux formes :

- Une version récursive à gauche et non factorisée qui se prête mieux à la réflexion.
- Une version LL(3) qui est la seule supportée par EGG.

Pour la transformation de la sémantique associée à l'élimination de la récursivité à gauche, et à la factorisation se replonger dans le cours et le TD correspondant.

Listing 1: Grammaire MJAVA.syn

```
1  ------------------------------------------------------------
   -- PROJET3 STL 07-08 - micro java : grammaire
   -- Version Recursive Gauche & Non factorisée

   PROGRAMME -> IMPORTS DEFCLASSE
6  -- importation
   IMPORTS ->
   IMPORTS -> import ident pv IMPORTS
   -- definition d'une classe
   DEFCLASSE -> classe ident EXTENSION aco DEFS acf
11 -- heritage
   EXTENSION -> etend ident
   EXTENSION ->
   -- les attributs/methodes
   DEFS ->
16 DEFS -> DEFQUAL DEF DEFS
   -- public, private, protected, static
   DEFQUAL ->
   DEFQUAL -> public DEFQUAL
   DEFQUAL -> prive DEFQUAL
21 DEFQUAL -> protect DEFQUAL
   DEFQUAL -> static DEFQUAL
   -- attribut
   DEF -> TYPE ident pv
   -- methode (fonction)
26 DEF -> TYPE ident paro PARFS parf BLOC
   -- methode (procedure)
   DEF -> void ident paro PARFS parf BLOC
   -- constructeur
   DEF -> ident paro PARFS parf BLOC
31 -- les types
   TYPE-> int
   TYPE-> bool
   TYPE-> ident
   -- parametres de methodes
36 PARFS ->
   PARFS -> PARF PARFSX
   PARFSX ->
   PARFSX -> virg PARF PARFSX
   PARF -> TYPE ident
41 -- corps de methode et bloc d'instructions
   BLOC -> aco INSTS acf
   -- instructions
   INSTS ->
   INSTS -> INST INSTS
46 -- declaration de variable locale avec ou sans init
   INST-> TYPE ident AFFX pv
   -- instruction expression
   INST -> E pv
   -- bloc d'instructions
51 INST -> BLOC
   -- conditionnelle
   INST -> si paro E parf BLOC SIX
   SIX -> sinon BLOC
   SIX ->
56 -- return
   INST -> retour E pv
   -- tant que
   INST -> tantque paro E parf BLOC
```

```
    -- les expressions
61  -- affectation
    E -> ER affect ER
    E -> ER
    -- relation
    ER -> ES OPREL ES
66  ER -> ES
    -- les operateurs rel
    OPREL -> inf
    OPREL -> infeg
    OPREL -> sup
71  OPREL -> supeg
    OPREL -> eg
    OPREL -> neg
    -- addition, ...
    ES -> ES OPADD T
76  ES -> T
    -- operateurs additifs
    OPADD -> plus
    OPADD -> moins
    OPADD -> ou
81  -- multiplication, ...
    T -> T OPMUL F
    T -> F
    -- operateurs mul
    OPMUL -> mult
86  OPMUL -> div
    OPMUL -> mod
    OPMUL -> et
    -- expressions de base
    F -> entier
91  F -> vrai
    F -> faux
    -- null
    F -> nil
    F -> paro E parf
96  -- new
    F -> nouveau TYPE paro ARGS parf
    -- unaire
    F -> OPUN F
    OPUN -> plus
101 OPUN -> moins
    OPUN -> non
    F -> FQ
    -- acces attribut d'un objet
    FQ -> FQ pt ident
106 -- appel methode sur objet
    FQ -> FQ pt ident paro ARGS parf
    -- acces methode de this
    FQ -> ident paro ARGS parf
    -- acces variable locale ou attribut de this
111 FQ -> ident
    -- this
    FQ -> this
    -- appel super
    FQ ->  super
116 -- liste d'arguments
    ARGS ->
    ARGS -> E ARGSX
    ARGSX ->
    ARGSX -> virg E ARGSX
```

Traduction des Langages

Examen 2006-2007. Durée 1H30. Documents autorisés.

1 Grammaire attribuée

Soit le fragment de grammaire attribuée :

```
I  -> if E then IS
                    {
                    E.val = faux;
                    E.etiquette = I.suivant;
                    IS.suivant = I.suivant;
                    }
IS -> I
                    {I.suivant = IS.suivant; }
IS -> IS  I
                    {
                    IS1.suivant = GenEtiq();
                    I.suivant = IS.suivant;
                    }
E  ->  E Opbool E
                    {
                    E2.etiquette = E.etiquette;
                    E2.val = E.val;
                    si Opbool.op = "ORELSE" alors
                       E1.val = vrai
                       si E.val alors
                           E1.etiquette = E.etiquette
                       sinon
                           E1.etiquette = GenEtiq();
                    sinon
                       E1.val = faux;
                       si E.val alors
                           E1.etiquette = GenEtiq();
                       sinon
                           E1.etiquette = E.etiquette
                     }
E  -> true
E  -> false
Opbool -> orelse
                    {Opbool.op = "ORELSE"; }
Opbool -> andthen
                    {Opbool.op = "ANDTHEN"; }
```

(Pour information seulement, cette grammaire permet d'optimiser la génération de code pour les expressions booléennes).

1. Sans le justifier, donner la nature des attributs sémantiques de cette grammaire attribuée.
2. Les valeurs des attributs sémantiques peuvent-elles calculées dans une analyse descendante gauche-droite ? Pourqoui ?

2 BLOC : Type Pile

- La syntaxe de BLOC est fournie en annexe mais on suppose connue la sémantique par attributs de la gestion de la table des symboles, du contrôle de type, et de la génération de code TAM du langage BLOC qui n'est donc pas fournie ici.
- Toute réponse devra être justifiée.
- Il est conseillé de lire tout le texte avant de répondre, pour avoir une vision globale du problème.

On veut ajouter à BLOC le type Pile, et ses opérations habituelles :

- push(p, v)
- pop(p)
- isEmpty(p)

On ajoute donc la règle de production suivante :

```
5) TYPE -> TYPE stack
```

1. Ajouter les règles de production nécessaires pour qu'un programme puisse manipuler une pile. On ne cherchera pas à obtenir une grammaire LL(1).
2. Donner un exemple simple d'un programme manipulant une pile.
3. Enumérer les ajouts/modifications à apporter au compilateur de BLOC pour prendre en compte ce nouveau type.
4. Modifier/Compléter la grammaire attribuée du contrôle de types pour prendre en compte le type Pile et sa manipulation.
5. Pour mettre en évidence la gestion de la mémoire associée à la manipulation d'une pile, dessiner l'état de la mémoire TAM (pile ?, tas ?) à des moments choisis de l'exécution de votre programme.
6. Donner le code TAM associé à votre exemple.
7. Modifier/Compléter la grammaire attribuée de la génération de code TAM.

A Syntaxe initiale de Bloc

```
1)  PROG    -> ident BLOC
2)  BLOC    -> { INSTS }
3)  TYPE    -> bool
4)  TYPE    -> int
6)  INSTS   -> ^
7)  INSTS   -> INST INSTS
8)  INST    -> TYPE ident = TERME ;
9)  INST    -> ident = TERME ;
10) INST    -> if ( TERME ) BLOC else BLOC
11) INST    -> while ( TERME ) BLOC
12) INST    -> print ( TERME ) ;
13) TERME   -> FACTEUR
14) FACTEUR -> ident
15) FACTEUR -> entier
16) FACTEUR -> vrai
17) FACTEUR -> faux
18) FACTEUR -> ( TERME )
```