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 d'autres assembleurs avec le minimum de changement.

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

1. L'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

NB. Ces concepts composent le 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 (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 un assembleur simple pour la génération, mais essayez d'être le plus générique possible pour éventuellement traiter d'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 vendredi 20 novembre 2009 et ne devra comporter que des membres de votre groupe de TD. Vous me préviendrez par mail : 'marcel.gandriau@enseeiht.fr'

Le projet se terminera le mercredi 18 janvier 2010. Les tests auront lieu le même jour de 14h à 18h.

Les sources (projet Eclipse et version 'make'), la documentation (TAM, EGG) et le présent sujet sont dans /mnt/n7fs/ens/gen6/0910.

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 09-10 - micro java : grammaire
   -- Version Recursive Gauche & Non factorisée

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

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

Listing 2: Grammaire MJAVA.egg

```
-- PROJET3 STL 09-10 - micro java : grammaire
option auto= true;
option version = 0.0.0 ;
option k=3;

-- les attributs semantiques
-- les terminaux

space separateur is  "[\r\n\t ]+";
space comm is  "\/\/[^\n]*\n";
sugar import is  "import";
sugar paro is  "\(";
sugar parf is  "\)";
sugar aco is  "\{";
sugar acf is  "\}";
sugar cro is  "\[";
sugar crf is  "\]";
sugar virg is  ",";
sugar pv is  "\;";
sugar pt is  "\.";
sugar affect is  "=";
sugar si is  "if";
sugar sinon is  "else";
sugar tantque is  "while";
sugar void is  "void";
sugar int is  "int";
sugar bool is  "boolean";
sugar classe is  "class";
sugar etend is  "extends";
sugar public is  "public";
sugar private is  "private";
sugar retour is  "return";
sugar nouveau is  "new";
sugar null is  "null";
sugar inf is  "\<";
sugar infeg is  "\<=";
sugar sup is  "\>";
sugar supeg is  "\>=";
sugar eg is  "==";
sugar neg is  "\!=";
sugar plus is  "\+";
sugar moins is  "\-";
sugar ou is  "\|\|";
sugar mult is  "\*";
sugar div is  "\/";
sugar mod is  "\%" ;
sugar et is  "\&\&";
sugar non is  "\!";
sugar vrai is  "true";
sugar faux is  "false";
term entier is  "[0-9]+";
term ident is  "[_A-Za-z][_0-9A-Za-z]*";

---------- REGLES DE PRODUCTION
    ----------------------------------------

PROGRAMME -> IMPORTS DEFCLASSE;
IMPORTS -> ;
IMPORTS -> IMPORT IMPORTS  ;
IMPORT -> import ident pv ;
```

```
-- definition d'une classe
DEFCLASSE -> classe ident EXTENSION aco DEFS acf ;
-- heritage
EXTENSION -> etend ident ;
EXTENSION -> ;
-- les attributs
DEFS ->   ;
DEFS ->  public DEF DEFS   ;
DEFS ->  private DEF DEFS   ;
-- attribut
DEF ->  TYPE ident pv  ;
-- methode (fonction)
DEF ->  TYPE ident paro PARFS parf BLOC  ;
-- methode (procedure)
DEF ->  void ident paro PARFS parf BLOC  ;
-- constructeur
DEF -> ident  paro PARFS parf BLOC  ;
-- les types
TYPE-> int  ;
TYPE-> bool ;
TYPE-> ident ;
-- parametres de methodes
PARFS ->  ;
PARFS ->  PARF PARFSX ;
PARFSX ->  ;
PARFSX -> virg  PARF  PARFSX ;
PARF ->  TYPE ident  ;
-- corps de methode et bloc d'instructions
BLOC ->  aco INSTS acf  ;
-- instructions
INSTS ->   ;
INSTS ->  INST INSTS  ;
-- declaration de variable locale avec ou sans init
INST->  TYPE ident AFFX pv ;
-- instruction expression
INST ->  E pv ;
-- bloc d'instructions
INST ->  BLOC  ;
-- conditionnelle
INST ->  si paro E parf BLOC SIX ;
SIX ->   sinon BLOC ;
SIX ->;
-- return
INST ->  retour E pv ;
-- tant que
INST ->  tantque paro E parf BLOC ;
-- les expressions
E ->  ER AFFX ;
-- affectation
AFFX ->  affect ER  ;
AFFX -> ;
-- relation
ER ->   ES ERX ;
ES ->  T ESX ;
ERX ->  OPREL ES ;
ERX -> ;
OPREL -> inf ;
OPREL -> infeg ;
OPREL -> sup ;
OPREL -> supeg ;
OPREL -> eg ;
```

```
    OPREL -> neg ;
    -- addition, ...
    ESX ->   OPADD T ESX ;
    ESX ->;
124 OPADD -> plus ;
    OPADD -> moins ;
    OPADD -> ou ;
    T ->  F TX ;
    -- multiplication, ...
129 TX ->   OPMUL F TX ;
    TX -> ;
    OPMUL -> mult ;
    OPMUL -> div ;
    OPMUL -> mod ;
134 OPMUL -> et ;
    -- expressions de base
    F -> entier  ;
    F -> vrai  ;
    F -> faux  ;
139 -- unaire
    F ->  OPUN F ;
    OPUN -> plus;
    OPUN -> moins;
    OPUN -> non;
144 -- null
    F -> null  ;
    F ->  paro E parf ;
    -- new
    F -> nouveau  TYPE paro ARGS parf  ;
149 F ->  ident Q ;
    -- Q = qualificateur de variable
    Q ->  ;
    -- acces attribut
    Q ->  pt ident  Q ;
154 -- arguments d'appel de methode
    Q ->  paro ARGS parf Q ;
    ARGS ->  E ARGSX  ;
    ARGS -> ;
    ARGSX ->   virg E ARGSX  ;
159 ARGSX -> ;

    end
```

Dans cette grammaire les actions sémantiques associées à l'axiome initialisent une variable 'machine' qui peut-etre transmise (sous la forme d'un attribut sémantique) aux autres symboles pour faciliter la génération de code. En effet cet attribut sera une instance de la classe AbstractMachine qui fournit les caractéristiques du processeur choisi (TAM, X86, SPARC) ainsi que les fonctions de la génération de code. Par exemple :

- Noms des registres de données ou d'adresse
- Code pour la réservation de place dans la pile
- Code pour l'affectation d'une zone mémoire
- Code pour l'allocation et libération de registres
- Code pour l'appel de fonction
- ...