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. La définition d'une classe et du type associé
2. L'héritage et le sous-typage associé
3. Quelques opérations arithmétiques et booléennes
4. L'accès aux attributs et méthodes d'une instance de classe
5. L'appel de méthodes par liaison tardive

NB. Ces concepts sont le minimum à réaliser.

NB2. Toutes les classes à compiler seront dans un même fichier.

NB3. Dans le fichier à compiler, les classes seront supposées définies avant d'être utilisées (pas de référence en avant).

NB4. Dans une classe, les attributs et les méthodes seront supposés définis avant d'être utilisés (pas de référence en avant).

NB5. La redéfinition d'une méthode dans une sous-classe doit être possible, mais la surcharge d'une méthode n'est pas à prendre en compte.

NB6. L'accès à un attribut ou méthode sera supposé public.

2 Moyens et conseils

Le compilateur sera écrit en utilisant Java et le générateur de compilateur EGG étudié en TD et TP. L'utilisation d'Eclipse doit permettre 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, ...), paramètres, 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. Il est important de bien prendre en compte tous les besoins lors de sa conception car 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 il demandé 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, (par mail, en séance de suivi) si vous avez des doutes sur votre conception.

3 Dates, Remise

Le projet a commencé ...

Votre trinome doit être constitué avant le lundi 13 mai 2013 8h. Vous me préviendrez de la composition du trinôme par mail à l'adresse

marcel.gandriau@enseeiht.fr

Les tests auront lieu le mardi 11 juin 2013 de 14h à 17h.

Les sources (projet Eclipse et version 'make'), la documentation (TAM, EGG) et le présent sujet sont sur Moodle.

Une archive de votre projet sera envoyée par mail à votre enseignant de TD. Cette archive contiendra :

- Les sources de votre projet ainsi que les fichiers de test.
- Un document (au format pdf uniquement) expliquant vos choix et limitations (ou extensions) dans le traitement de micro-java. Ce document ne sera pas long, mais le plus précis possible (schémas) pour nous permettre de comprendre et juger votre travail.

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 acceptée par EGG.

Pour la transformation de la sémantique associée à l'élimination de la récursivité à gauche, et à la factorisation, vous pouvez expoiter la transformation systématique étudiée en cours et en TD.

Listing 1: Grammaire MJAVA.syn

```
1  ------------------------------------------
   -- PROJET3 STL 11-12 - micro java : grammaire
   -- Version Recursive Gauche & Non factorisée

   PROGRAMME -> CLASSES
6  CLASSES ->
   CLASSES -> DEFCLASSE CLASSES
   -- definition d'une classe
   DEFCLASSE -> classe ident SUPER CORPS
   SUPER ->
11 SUPER -> extend ident
   CORPS -> aco DEFS acf
   -- les attributs
   DEFS ->
   DEFS ->  DEF DEFS
16 -- attribut
   DEF ->  TYPE ident pv
   -- methode (fonction)
   DEF ->  TYPE ident paro PARFS parf BLOC
   -- methode (procedure)
21 DEF ->  void ident paro PARFS parf BLOC
   -- constructeur
   DEF -> ident  paro PARFS parf BLOC
   -- les types
   TYPE-> int
26 TYPE-> bool
   TYPE-> ident
   -- parametres de methodes
   PARFS ->
   PARFS -> PARF PARFSX
31 PARFSX ->
   PARFSX -> virg PARF PARFSX
   PARF -> TYPE ident
   -- corps de methode et bloc d'instructions
   BLOC -> aco INSTS acf
36 -- instructions
   INSTS ->
   INSTS -> INST INSTS
   -- declaration de variable locale avec ou sans init
   INST-> TYPE ident pv
41 INST-> TYPE ident affect E pv
   -- instruction expression
   INST -> E pv
   -- bloc d'instructions
   INST -> BLOC
46 -- conditionnelle
   INST -> si paro E parf BLOC SIX
   SIX -> sinon BLOC
   SIX ->
   -- return
51 INST -> retour E pv
   -- les expressions
   -- affectation
   E -> ER affect ER
   E -> ER
56 -- relation
   ER -> ES OPREL ES
   ER -> ES
   -- les operateurs rel
   OPREL -> inf
```

```
OPREL -> infeg
OPREL -> sup
OPREL -> supeg
OPREL -> eg
OPREL -> neg
-- addition, ...
ES -> ES OPADD TERME
ES -> TERME
-- operateurs additifs
OPADD -> plus
OPADD -> moins
OPADD -> ou
-- multiplication, ...
TERME -> TERME OPMUL FACTEUR
TERME -> FACTEUR
-- operateurs mul
OPMUL -> mult
OPMUL -> div
OPMUL -> mod
OPMUL -> et
-- unaire
FACTEUR -> OPUN FACTEUR
OPUN -> plus
OPUN -> moins
OPUN -> non
-- expressions de base
FACTEUR -> entier
-- booleens
FACTEUR -> vrai
FACTEUR -> faux
-- null
FACTEUR -> null
FACTEUR -> F
-- expression parenthésée
F -> paro E parf
-- new
F -> nouveau TYPE paro ARGS parf
-- objet courant
F -> this
-- objet parent
F -> super
-- acces variable, parametre, ou attribut de this
F -> ident
F -> F pt ident
F -> F paro ARGS parf
-- liste d'arguments d'appel de methode
ARGS ->
ARGS -> E ARGSX
ARGSX ->
ARGSX -> virg E ARGSX
```

Listing 2: Grammaire MJAVA.egg

```
-- PROJET3 STL 12-13 - micro java : grammaire
option auto= true;
option version = 0.0.0 ;
option k=3;

-- les attributs semantiques
inh source : MJAVASourceFile for PROGRAMME;
-- les terminaux
space separateur is  "[\r\n\t ]+";
space comm is  "\/\/[^\n]*\n";
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 void is  "void";
sugar int is  "int";
sugar bool is  "boolean";
sugar classe is  "class";
sugar extend is  "extends";
sugar retour is  "return";
sugar this is  "this";
sugar super is  "super";
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 -> #init CLASSES #gen;
global
   machine : AbstractMachine;
#init {
local
do
   machine := PROGRAMME^source.getMachine();
end
```

```
}

#gen {
local
do
  call machine.writeCode(PROGRAMME^source.getFileName(), "");
end
}

CLASSES ->;
CLASSES -> DEFCLASSE CLASSES;
-- definition d'une classe
DEFCLASSE -> classe ident SUPER CORPS;
SUPER ->;
SUPER -> extend ident;
CORPS -> aco DEFS acf;
-- les attributs
DEFS ->;
DEFS ->  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 pv ;
INST-> TYPE ident affect E 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 ;
-- les expressions
E ->  ER AFFX ;
-- affectation
AFFX ->  affect ER  ;
AFFX -> ;
-- relation
```

```
    ER ->    ES ERX ;
    ES ->  TERME ESX ;
    ERX ->  OPREL ES ;
125 ERX -> ;
    OPREL -> inf ;
    OPREL -> infeg ;
    OPREL -> sup ;
    OPREL -> supeg ;
130 OPREL -> eg ;
    OPREL -> neg ;
    -- addition, ...
    ESX ->   OPADD TERME ESX ;
    ESX ->;
135 OPADD -> plus ;
    OPADD -> moins ;
    OPADD -> ou ;
    TERME ->  FACTEUR TX ;
    -- multiplication, ...
140 TX ->   OPMUL FACTEUR TX ;
    TX -> ;
    OPMUL -> mult ;
    OPMUL -> div ;
    OPMUL -> mod ;
145 OPMUL -> et ;
    -- unaire
    FACTEUR ->  OPUN FACTEUR ;
    OPUN -> plus;
    OPUN -> moins;
150 OPUN -> non;
    -- expressions de base
    FACTEUR -> entier;
    -- booleens
    FACTEUR -> vrai;
155 FACTEUR -> faux;
    -- null
    FACTEUR -> null;
    FACTEUR -> F;
    -- expression parenthesee
160 F -> paro E parf ;
    -- new
    F -> nouveau TYPE paro ARGS parf  FX;
    -- objet courant
    F -> this  FX;
165 -- objet parent
    F -> super  FX;
    -- acces variable, parametre, ou attribut de this
    F -> ident FX;
    FX -> ;
170 FX -> pt ident FX;
    -- appel methode sur objet
    FX -> paro ARGS parf FX;
    -- liste d'arguments d'appel de methode
    ARGS ->  E ARGSX  ;
175 ARGS -> ;
    ARGSX ->   virg E ARGSX  ;
    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
- ...