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

NB. Ces concepts sont le minimum à réaliser.

NB2. Toutes les classes compilées seront dans un même fichier.

NB3. Pour garder la grammaire assez simple, elle accepte des programmes qui n'ont éventuellement pas de sens. Vous pouvez sans perte de généralité supposer que vos programmes de tests n'en font pas partie. On peut citer :

- Implementation d'une interface par une autre interface
- Définition d'un corps dans une méthode d'une interface
- Définition d'attributs ou de constructeurs dans une interface.
- Appel d'un constructeur sur une interface
- new sur un type de base
- etc

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 et les interfaces (héritage, implantation, 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 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 2 décembre 2011 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 jeudi 26 janvier 2011. Les tests auront lieu le même jour de 8h à 12h.

Les sources (projet Eclipse et version 'make'), la documentation (TAM, EGG) et le présent sujet sont dans /mnt/n7fs/ens/gen6/1112.

Un document imprimé ou manuscrit (si votre écriture est suffisament lisible) 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 avant le test par mail 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 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 -> ENTITES
6  ENTITES ->
   ENTITES -> DEFCLASSE ENTITES
   ENTITES -> DEFINTERFACE ENTITES
   -- definition d'une classe
   DEFCLASSE -> classe ident SUPER CORPS
11 DEFINTERFACE -> interface ident SUPER CORPS
   -- 1 extends maximum suivi d'un implements maximum
   SUPER -> ETEND IMPL
   ETEND ->
   ETEND -> etend ident
16 IMPL ->
   IMPL -> implemente ident
   CORPS -> aco DEFS acf
   -- les attributs
   DEFS ->
21 DEFS ->  DEF DEFS
   -- attribut
   DEF ->  TYPE ident pv
   -- methode (fonction)
   DEF ->  TYPE ident paro PARFS parf CORPS
26 -- dans interface
   DEF ->  TYPE ident paro PARFS parf pv
   -- methode (procedure)
   DEF ->  void ident paro PARFS parf CORPS
   -- dans interface
31 DEF ->  void ident paro PARFS parf pv
   -- constructeur
   DEF -> ident  paro PARFS parf BLOC
   -- les types
   TYPE-> int
36 TYPE-> bool
   TYPE-> ident
   -- parametres de methodes
   PARFS ->
   PARFS -> PARF PARFSX
41 PARFSX ->
   PARFSX -> virg PARF PARFSX
   PARF -> TYPE ident
   -- corps de methode et bloc d'instructions
   BLOC -> aco INSTS acf
46 -- instructions
   INSTS ->
   INSTS -> INST INSTS
```

```
   -- declaration de variable locale avec ou sans init
   INST-> TYPE ident AFFX pv
51 -- instruction expression
   INST -> E pv
   -- bloc d'instructions
   INST -> BLOC
   -- conditionnelle
56 INST -> si paro E parf BLOC SIX
   SIX -> sinon BLOC
   SIX ->
   -- return
   INST -> retour E pv
61 -- les expressions
   -- affectation
   E -> ER affect ER
   E -> ER
   -- relation
66 ER -> ES OPREL ES
   ER -> ES
   -- les operateurs rel
   OPREL -> inf
   OPREL -> infeg
71 OPREL -> sup
   OPREL -> supeg
   OPREL -> eg
   OPREL -> neg
   -- addition, ...
76 ES -> ES OPADD T
   ES -> T
   -- operateurs additifs
   OPADD -> plus
   OPADD -> moins
81 OPADD -> ou
   -- multiplication, ...
   T -> T OPMUL F
   T -> F
   -- operateurs mul
86 OPMUL -> mult
   OPMUL -> div
   OPMUL -> mod
   OPMUL -> et
   -- expressions de base
91 F -> entier
   F -> vrai
   F -> faux
   -- null
   F -> null
96 F -> this
   F -> paro E parf
   -- new
```

```
    F -> nouveau TYPE paro ARGS parf
    -- unaire
101 F -> OPUN F
    OPUN -> plus
    OPUN -> moins
    OPUN -> non
    F -> FQ
106 -- acces attribut d'un objet
    FQ -> FQ pt ident
    -- appel methode sur objet
    FQ -> FQ pt ident paro ARGS parf
    -- acces methode de this
111 FQ -> ident paro ARGS parf
    -- acces variable locale ou attribut de this
    FQ -> ident
    -- liste d'arguments
    ARGS ->
116 ARGS -> E ARGSX
    ARGSX ->
    ARGSX -> virg E ARGSX
```

Listing 2: Grammaire MJAVA.egg

```
-- PROJET3 STL 11-12 - micro java : grammaire
option auto= true;
option version = 0.0.0 ;
option k=3;

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 interface is  "interface";
sugar etend is  "extends";
sugar implemente is  "implements";
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 -> ENTITES ;
ENTITES -> ;
ENTITES -> DEFCLASSE ENTITES  ;
ENTITES -> DEFINTERFACE ENTITES  ;
-- definition d'une classe
DEFCLASSE -> classe ident SUPER CORPS ;
DEFINTERFACE -> interface ident SUPER CORPS;
-- 1 extends maximum suivi d'un implements maximum
SUPER -> ETEND IMPL ;
ETEND -> ;
ETEND -> etend ident ;
IMPL -> ;
IMPL -> implemente ident ;
CORPS -> aco DEFS acf ;
-- les attributs
DEFS ->   ;
DEFS -> DEF DEFS   ;
-- attribut
DEF -> TYPE ident pv  ;
-- methode (fonction)
DEF -> TYPE ident paro PARFS parf MCORPS  ;
-- methode (procedure)
DEF -> void ident paro PARFS parf MCORPS  ;
-- dans classe ou dans interface
MCORPS -> pv ;
MCORPS -> 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 ;
102 SIX ->;
    -- return
    INST -> retour E pv ;
    -- les expressions
    E -> ER AFFX ;
107 -- affectation
    AFFX -> affect ER ;
    AFFX -> ;
    -- relation
    ER -> ES ERX ;
112 ES -> T ESX ;
    ERX -> OPREL ES ;
    ERX -> ;
    OPREL -> inf ;
    OPREL -> infeg ;
117 OPREL -> sup ;
    OPREL -> supeg ;
    OPREL -> eg ;
    OPREL -> neg ;
    -- addition, ...
122 ESX -> OPADD T ESX ;
    ESX ->;
    OPADD -> plus ;
    OPADD -> moins ;
    OPADD -> ou ;
127 T -> F TX ;
    -- multiplication, ...
    TX -> OPMUL F TX ;
    TX -> ;
    OPMUL -> mult ;
132 OPMUL -> div ;
    OPMUL -> mod ;
    OPMUL -> et ;
    -- expressions de base
    F -> entier ;
137 F -> vrai ;
    F -> faux ;
    -- unaire
    F -> OPUN F ;
    OPUN -> plus;
142 OPUN -> moins;
    OPUN -> non;
    -- null
    F -> null ;
    F -> paro E parf ;
147 -- new
```

```
    F -> nouveau  TYPE paro ARGS parf  ;
    F ->  ident Q ;
    -- Q = qualificateur de variable
    Q ->  ;
152 -- acces attribut
    Q ->  pt ident  Q ;
    -- arguments d'appel de methode
    Q ->  paro ARGS parf Q ;
    ARGS ->  E ARGSX  ;
157 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
- ...