Компілятор PL/IL

Проектування та розробка

Версія 2004.06.01

Максим Е. Сохацький (mes@ua.fm)
Олег В. Смірнов (_straycat@ukr.net)

Ітерація властива людині. Рекурсія – божественна.

Київський Політехнічний Інститут
2004

Зміст

Розділ 1

Вступ

1.1 Поставлені цілі

Основне завдання, поставлене перед розробниками цього проєкту, полягало в реалізації мови програмування PL/1 для єдиної мовної середовища Microsoft Common Language Runtime (CLR), з підтримкою Common Intermediate Language (CIL) – об'єктно-орієнтованого стекового асемблера для віртуальної машини CLR. Єдина мовна інфраструктура (Common Language Infrastructure) разом із системою класів Microsoft .NET Framework є найсучаснішим способом створення застосунків для операційних систем Microsoft.

1.2 Єдине мовне середовище Microsoft

Завдяки Єдиній системі типів (Common Type System, CTS) CLR забезпечує можливість використовувати однаковий спосіб взаємодії застосунків, написаних різними мовами. Такі технології пізнього зв'язування та компонентні моделі, як COM і ActiveX, що використовують домовленості щодо структур об'єктів, більше не потрібні. На сьогодні існує майже повний спектр імперативних мов програмування, які використовують CLR як носій об'єктного коду: Pascal, C, Java, Python, Perl, Cobol, Fortran. Реалізації бракує лише однієї з найстаріших і найпотужніших мов – PL/1, яку автори цієї роботи вирішили відродити в об'єктно-орієнтованій парадигмі.

1.3 Об'єктно-орієнтований PL/1

PL/1 досі залишається однією з найпотужніших мов програмування завдяки своїй здатності підставляти константи, описані в програмі, як виконувану програму. Це одна з багатьох переваг PL/1, яка легко реалізується в середовищі CLR.

1.4 Особливості PL/IL

Компілятор PL/IL, який використовує CLR і .NET Framework як хост, має низку особливостей:

- Компіляція EXE та DLL збірок CLR.
- Використання класів .NET Framework.
- Єдина система типів CTS.
- Генерація лістингу програми на IL.
- Використання Reflection для підставок.
- Структурна обробка винятків.
- Вбудований контроль типів.

Завдяки цим можливостям можна створювати навіть вебсервіси, використовуючи PL/IL. Крім того, можна використовувати збірки, написані іншими мовами, що підтримують CLI, наприклад C# або Visual Basic .NET, через лінкування або динамічне завантаження. Діалект PL/IL максимально наближений до PL/M – оптимізованої версії PL/1 для мікроконтролерів.

1.5 Перспективи

На відміну від віртуальних машин Java, CLR має специфікацію на єдину систему типів, спільну для всіх мов CLI. Крім того, на відміну від Java, усі специфікації на ці технології затверджені ECMA, що гарантує сумісність майбутніх кросплатформних реалізацій CLR, таких як MONO.

Розділ 2

Теоретичні відомості

2.1 Структура компілятора

Основні етапи, які проходить компілятор, включають:

- Лексичний аналіз.
- Синтаксичний аналіз.
- Контекстний аналіз.
- Машинно-незалежний оптимізатор.
- Генератор коду.
- Оптимізатор коду.

Інформація, необхідна на етапах компіляції, класифікується так:

- Вихідний текст програми.
- Таблиця термінальних символів мови.
- Лексична згортка.
- Правила граматики.
- Дерево виведення.
- Абстрактна програма.
- Проміжний код.
- Об'єктний код.

2.2 Теоретичні питання

Можна виділити такі теми для освоєння теорії побудови програмних систем:

1. Лексичний аналіз.
2. Синтаксичний аналіз.
3. Синтаксично керована трансляція.
4. Система типів і середовище виконання.
5. Генерація проміжного коду.
6. Генерація об'єктного коду.

2.3 Основні методи синтаксичного аналізу

Нисхідний аналіз

Усунувши з граматики ліву рекурсію, можна ефективно реалізувати аналіз методом рекурсивного спуску без відкатів. Також можна використовувати табличні методи аналізу без відкатів для граматик із усунутою лівою рекурсією. Автори обрали нисхідний аналіз, оскільки LL-граматика виглядає більш естетично. Метод рекурсивного спуску без відкатів полегшує розуміння компілятора іншими розробниками, підвищує ступінь спільної розробки та є більш гнучким і розширюваним порівняно з табличними методами.

Висхідний аналіз

Для висхідного аналізу існують алгоритми типу «перенесення-згортка», а також генератори аналізаторів LR і LALR граматик.

Метод рекурсивного спуску

Цей метод добре підходить для простих LL(1)-граматик, хоча може використовуватися і для синтаксичних аналізаторів із відкатами. Синтаксичний аналізатор будується так, що для кожного нетермінала визначається одна рекурсивна процедура. Правила граматики перетворюються для однозначного визначення альтернатив і цілеспрямованого виведення. Наприклад, якщо є правила:

$$A \to \alpha\, t_1\, \beta \mid \alpha\, t_2\, \gamma\,;\ t_1, t_2 \in \Sigma\,;\ \alpha, \beta, \gamma - \text{довільні ланцюжки},$$

вони перетворюються до виду:

$$A \to \alpha\, (\, t_1\, \beta \mid t_2\, \gamma\,),$$

де $(\dots)$ – скобки факторизації.

Правила виду:

$$A \to \alpha \mid A\, t\, \beta,$$

де $t \in \Sigma$, $\alpha, \beta \in (N \cup \Sigma)^*$, перетворюються до:

$$A \to \alpha \{ t \beta \},$$

де $\{\dots\}$ – рекурсивна частина.

Правила виду:

$$A \to \alpha \mid \alpha t \beta,$$

записуються як:

$$A \to \alpha [t \beta],$$

де $[\dots]$ – необов'язкова конструкція.

Приклад граматики:

$$\begin{aligned}
&A_1 \to A_2 := A_3 \mid \text{if } A_3 \text{ then } A_1 \mid \text{if } A_3 \text{ then } A_1 \text{ else } A_1 \\
&A_2 \to i \mid i (A_3) \\
&A_3 \to A_4 \mid A_3 + A_4 \\
&A_4 \to A_5 \mid A_4 * A_5 \\
&A_5 \to A_2 \mid (A_3)
\end{aligned}$$

Для процедур правила переписуються так:

$$\begin{aligned}
&A_1 \to A_2 := A_3 \mid \text{if } A_3 \text{ then } A_1 [\text{else } A_1] \\
&A_2 \to i [(A_3)] \\
&A_3 \to A_4 \{ + A_4 \} \\
&A_4 \to A_5 \{ * A_5 \} \\
&A_5 \to A_2 \mid (A_3)
\end{aligned}$$

Приклад процедури для A_1:

```
1  int next;
2  void A1() {
3      if (next == IF) {
4          scan();
5          A3();
6          if (next != THEN) error();
7          else {
8              scan();
9              A1();
10             if (next == ELSE) {
11                 scan();
12                 A1();
13             }
14         }
15     } else {
16         A2();
17         if (next != AS) error();
18         else {
```

```
19                scan();
20                A3();
21            }
22        }
23    }
```

Розділ 3

Компілятор PL/IL

3.1 Історія

PL/IL – це підмножина мови PL/1, створеної IBM, яка активно використовувалася в 60-х роках. У країнах колишнього СРСР ця мова застосовувалася в машинах ЄС ЕОМ, що було логічно, оскільки ЄС ЕОМ була аналогом обчислювальних комплексів IBM System/360. Символ «/» характерний для назв продуктів IBM, наприклад, System/360, ES/9000, RS/6000, AS/400, PS/2, OS/2, PL/1.

Від часу створення PL/1 було реалізовано багато мов із повністю або частково сумісним синтаксисом. Компанія Intel розробила PL/M – підмножину PL/1 для мікроконтролерів, що відображено в «M» у назві. PL/IL є підмножиною PL/M, яка, у свою чергу, є підмножиною PL/1. Наразі підмножина PL/SQL використовується в системах управління базами даних Oracle. Багато компаній, що займаються підтримкою старих мейнфреймів IBM, пропонують компілятори PL/1 для нових процесорів.

Як результуючий асемблер обрано об'єктну стекову машину Microsoft .NET CLR – Microsoft Intermediate Language (MS IL), що й дало назву «PL/IL».

3.2 Діаграма компілятора

Рис. 1. Структура компілятора (зображення не включено)

Рис. 3.1: Структура компілятора

3.3 Алфавіт мови

Лексичний аналізатор LEX обробляє коментарі, числові константи, ідентифікатори та виконує пошук ідентифікаторів. Односимвольні термінали:

$$\$, =, -, +, , /, ', :, ;, ., , , (,), \%, \&, ' \#, |, !, >, <, \sim, `, \backslash,], [, \}, \{, ?, \_$$

Двосимвольні термінали:

$$<=, **, >=, ! =, - >, /*, */, ! <, ! >, ||, ..$$

Операції-мнемоніки:

GT, LT, GE, LE, NG, NL, NE, EQU, CAT, PT, NOT, OR, AND, XOR, MOD, DIV, MUL, PLUS, M

Ключові слова:

ASSIGN, BY, CALL, DECLARE, DCL, DO, ELSE, END, GO, GOTO, IF, INITIAL, LABEL, LITER

3.4 Граматика мови

Граматика без лівої рекурсії базується на граматиці мови MyC із прикладів Microsoft Visual Studio .NET, модифікованій для PL/1:

```
1  letter                  ::= "A-Za-z";
2  digit                   ::= "0-9";
3  name                    ::= letter { letter | digit };
4  integer                 ::= digit { digit };
5  ident                   ::= name | label;
6  factor          ::= (ident | integer | "(" expr ")" );
7  unary_factor    ::= ["+"|"-"] factor;
8  term1                   ::= ["*"|"/"] factor;
9  term0                   ::= factor { term1 };
10 first_term              ::= unary_factor term1;
11 math_expr               ::= first_term { ["+"|"-"] term0 }
12 rel_expr                ::= math_expr
       ("=="|"!="|"<"|">"|">="|"<=")
13     math_expr;
14 not_factor              ::= ["!"] rel_expr;
15 term_bool               ::= not_factor { ("&" | "&&")
       not_factor };
16 bool_expr               ::= term_bool { ("|" | "^")
       term_bool };
17 expr                    ::= bool_expr;
18 var                     ::= ident;
19 var_list                ::= var { "," var_list };
20 assign_stmt             ::= var_list "=" expr;
21 param                   ::= "(" var_list ")";
22 declare                 ::= ( "dcl" | "declare" ) "(" var |
23     param ")" { "(" integer ")" } type;
```

```
type                    ::= "fixed"|"binary"|"float"|"
    decimal" ;
if_stmt                 ::= "if" expr "then" stmt [ "else"
    stmt;
do_stmt                 ::= "do" (
null_stmt  smtm; |
"while" expr null_stmt stmt |
assign_stmt  "to" expt null_stmt stmt
) "end" null_stmt
             rets_stmt          ::= "returns" "(" type ")";
proc_decl               ::= ident ":" proc ;
proc                    ::= param ["recursive"][ rets_stmt ]
                        null_stmt stmt { stmt }
    "end" ident  null_stmt ;
goto_stmt               ::= ("go to" | "goto" ) ( ident |
    integer )
    null_stmt;
ident_stmt              ::= (ident "=" expr| ident ":" stmt
    | ident { , ident } = expr) null_stmt;
ret_stmt                ::= "return" [ expr ] ";";
null_stmt               ::= ";";
stmt                    ::= (ret_stmt null_stmt | ident_stmt
    |
do_stmt null_stmt | goto_stmt null_stmt | proc_decl
    null_stmt |
decl_stmt null_stmt | if_stmt );
program_stmt    ::= stmt { stmt };
```

(Далі граматика продовжується для змінних, присвоєнь, процедур тощо, як у вихідному документі.)

3.5 Опис мови

Ідентифікатори користувача

Ідентифікатор обмежений довжиною 32 символи, повинен починатися з літери, може містити літери, цифри та знак долара, який ігнорується компілятором. Ідентифікатори нечутливі до регістру.

Зарезервовані слова

Зарезервовані слова завжди у верхньому регістрі:

ASSIGN, BY, CALL, DECLARE, DCL, DO, ELSE, END, GO, GOTO, IF, INITIAL, LABEL, LITERALLY, OPT

Константи

Числові константи починаються з цифри і можуть мати специфікатори: O (восьмеричні), Q (восьмеричні), H (шістнадцяткові), D (десятичні), B

(бінарні). Без специфікатора – десятковий режим. Комплексні константи та константи з плаваючою чи фіксованою комою не підтримуються.

Операції у виразах

Розділювачі: «(», «)». Арифметичні операції: $+, -$, PLUS, MINUS, $*, /$, MOD. Операції відношень: $<, >=, >, <=, =, !=$. Булеві операції: NOT, AND, OR, XOR.

Пріоритети операцій

1. $*, /$, MOD
2. $+, -$, PLUS, MINUS
3. $<, <=, =, =>, >, !=$
4. NOT
5. AND
6. OR, XOR

Мовні конструкції

Приклади:

```
IF ((4D > 5 * (-2AH) / (-5+7)) & (6 < 9)) THEN ;
DO WHILE expression; { statement; } END;
```

3.6 Емітер

Підсистема генерації коду EMT має модульну архітектуру. Під час контекстно-керованого емітінгу будується дерево виведення IAsm, яке використовується для генерації об'єктного коду.

3.7 Повідомлення компілятора

Приклади помилок:

- PL0101: invalid typecast (float to int)
- PL0201: invalid octal constant
- PL0301: unhandled instruction type

Табл. 3.1: Приведення типів при цілочисельних операціях

	Int16	Int32	Int64	Float	double
Int16	Int16	Int32	Int64	Float	double
Int32	Int32	Int64	Int64	Float	double
Int64	Int64	Int64	Int64	float	double
Float	float	float	float	Float	double
double	double	double	double	double	double

3.8 Автоматичне приведення скалярних типів

3.9 Бібліотека вбудованих функцій

Приклади: `System.Console.WriteLine`, `System.Console.ReadLine`.

3.10 Асемблер

Список інструкцій CLI (з ECMA):

```
0x00 nop
0x01 break
0x02 ldarg.0
...
```

Розділ 4

Тестування

4.1 Системи автоматичного тестування

Для перевірки стабільності використано систему NUNIT.

4.2 Розробка штурмових тестів

Штурмові тести розроблялися з особливою ретельністю. Приклади програм:

```
Main: PROC;
    DCL ( P1, P2, P3 ) FIXED;
    P1 = 1Ah;
    P2 = 1001b;
    P3 = 150;
    CALL PRINT_I(P1);
    CALL PRINT_I(P2);
    CALL PRINT_I(P3);
END Main;
```

Розділ 5

Епілог

5.1 Висновки

Реалізація компілятора за допомогою сучасних засобів розробки не становить труднощів. На Java чи C це було б менш привабливо.

5.2 Подяки

Дякуємо науковим керівникам та розробникам Microsoft за технічні засоби.

5.3 Додатки

Література

1. Фролов Г.Д., Олюнін В.Ю. Практичний курс програмування ПЛ/1.
2. Ахо, Сеті, Ульман. Компілятори.
3. Дональд Кнут. Мистецтво програмування. 3 томи.
4. MSDN April 2003.