Системы программирования.

лекции 18.04.2011

Генерация внутреннего представления.
ПОЛИЗ.
Оптимизация.

(С) Корухова Ю.С., 2011

Этапы трансляции

- лексический анализ
- синтаксический анализ
- семантический анализ
- генерация внутреннего представления программы
- оптимизация
- генерация объектной программы

Основные свойства языка внутреннего представления программ

– Язык позволяет фиксировать синтаксическую структуру исходной программы

– Текст на этом языке можно генерировать во время синтаксического анализа

– Конструкции языка должны достаточно просто транслироваться в объектный код либо достаточно эффективно интерпретироваться

Привести пример ошибки, которая может быть обнаружена именно на этапе интерпретации внутреннего представления программы

ПОЛИЗ
(ПОЛьская Инверсная Запись)

Инфиксная запись	ПОЛИЗ (постфиксная запись)
Операнд X (переменная, константа)	X
Выражение op E (op — унарная операция, E - выражение)	E' op (E' — ПОЛИЗ E)
Выражение E1 op E2 (op — бинарная операция, E1, E2 - выражения)	E1' E2' op (E1' и E2' — ПОЛИЗ E1 и E2)
Выражение (E) (E — выражение)	E' (E' — ПОЛИЗ E)

Интерпретация ПОЛИЗа

Будем использовать стек.
Просматриваем ПОЛИЗ поэлементно слева направо

(1) если очередной элемент — операнд,
то его значение заносится в стек;
(2) если очередной элемент — операция,
из стека извлекаются ее операнды, выполняется операция, результат снова заносится в стек;
(3) когда выражение, записанное в ПОЛИЗе, прочитано, в стеке останется один элемент — это значение всего выражения.

Примеры:

```
a b or c and          -------->    (a or b) and c

a b # -               -------->    a-(-b)
или a 0 b - -
```

по обозначению операции в ПОЛИЗе должно быть известно, сколько у нее операндов

Генерация ПОЛИЗа

Будем генерировать ПОЛИЗ методом синтаксически управляемого перевода.

Входной язык L1 = { программы }
Целевой язык L2 = { внутренние представления программ}

Построение внутреннего представления делается процедурами РС (одновременно с синтаксическим анализом)

Для сгенерированного таким методом ПОЛИЗа

1. **Сохраняется порядок операндов**
2. **Для некоторых операций потребуется изменение обозначений**
3. **Потребуется ввести дополнительные операции**

ПОЛИЗ выражений

Обозначения
c- анализируемая лексема
c.value — номер строки в таблице, где записана лексема
TD — таблица разделителей (массив строк)
x — в каждой из функций — локальная переменная

E → E1 [= | < | > | <= | >= | !=]<x=c.value;> E1 <cout <<TD[x];> | E1
E1 → T {[+ | - | *or*] <x=c.value;> T <cout <<TD[x];>}
T → F {[* | / | *and*] <x=c.value> F <cout <<TD[x];>}
F → I <печать идентификатора> |
N <печать константы>|
L |
***not* F <cout <<" not ";> |**
(E)
L → true <cout <<" true ";> | false <cout <<" false ";>

ПОЛИЗ операторов

В отличие от выражений, после интерпретации ПОЛИЗа оператора, в стеке НЕ остается результата вычислений.

Дополнительные операции

;

p !

E p !F

Оператор – выражение, после которого стоит ;

S → E;

E' ;

Операции ввода и вывода

S → read(I); | write(E);

I read E' write

Оператор присваивания

S → I := E

I E' := ;

ПОЛИЗ операторов (продолжение)

Оператор безусловного перехода

S → goto M;

<номер элемента ПОЛИЗа> !

Условный оператор

S → if E then S

if (not E) goto 1;
S
1:

S → if E then S else S

if (not E) goto 1;
S_1
goto 2;
1: S_2
2:

ПОЛИЗ операторов (продолжение)

Операторы цикла

с предусловием

S → while (E) do S

с постусловием

S → do S while (E);

цикл for

S → for (E;E;E) S

Оператор выбора

S → switch (E) {
case E: {S}
{case E: {S}}
}

Другие способы внутреннего представления

1. Связные списочные структуры

2. Многоадресный код с явно именуемыми результатами
тетрады: операция операнд1 операнд2 результат

1	*	B	C	T1
2	+	T1	D	T2
3	*	B	10	T3
4	-	T2	T3	T4
5	=	T4	<пусто>	A

3.Многоадресный код с неявно именуемым результатом
триады: операция операнд операнд

1	*	B	C
2	+	^1	D
3	*	B	10
4	-	^2	^3
5	=	A	^4

Этапы трансляции

- лексический анализ
- синтаксический анализ
- семантический анализ
- генерация внутреннего представления программы
- оптимизация
- генерация объектной программы

Оптимизация программ

Обработка, связанная с переупорядочением и изменением операций в транслируемой программе в целях получения более эффективной объектной программы.

Параметры оптимизации: скорость выполнения, объем памяти, равномерная загрузка оборудования многопроцессорного комплекса, …

При оптимизации
1. преобразования должны сохранять семантику программы (т.е. быть эквивалентными);

2.стоимость преобразования должна быть сопоставима с затрачиваемыми усилиями и побочными эффектами;

3.в результате преобразований работа программ «в среднем» должна улучшаться (почти на всех допустимых данных), лишь на некоторых (редко встречающихся) может быть ухудшение характеристик

Оптимизация программ

Машинно-независимая оптимизация

проводимые в рамках этого процесса преобразования не зависят от архитектуры вычислительной системы, для которой предназначена объектная программа

Машинно-зависимая оптимизация

преобразования ориентированы на архитектуру конкретной вычислительной системы, то есть на совокупность программных и аппаратных составляющих и взаимосвязи между ними. Обычно преобразуется объектная программа / внутреннее представление

Машинно- независимая оптимизация

Оптимизация линейных участков программы

– вычисление константных выражений

```
A := sin (2 * 3.1415 * B + C); // B, C – константы
#define Pi 3.1415926
A := sin (2 * Pi * B+C );
```

– арифметические преобразования

A=B*C+B*D -----> A=B*(C+D)

– устранение общих подвыражений (избыточных вычислений)

– удаление ненужных присваиваний ("распространение копий") и других операций

– перестановка независимых смежных участков программ

A := 2 * B * 3 *C; -----> A := (B * C) * (2 * 3);

– оптимизация вычисления логических выражений

– удаление недостижимых участков программы

Машинно- независимая оптимизация (прод.)

Оптимизация передачи параметров и вызовов функций

– прямая подстановка тел функций в основной текст программы

– передача параметров через глобальные переменные, которые впоследствии связываются с регистрами центральных процессоров (машинно-зависимая оптимизация!)

Машинно- независимая оптимизация (прод.)

Оптимизация циклов

Цикл — любая последовательность операторов, которая может выполняться повторно.
(Не обязательно оформлена как оператор цикла!)

– вынесение инвариантных вычислений

– замена операции с переменными цикла

– слияние, расщепление и развертывание циклов

Машинно- зависимая оптимизация

– учет регистровой структуры вычислительной аппаратуры (оптимальное распределение регистров, передача параметров в процедуры и функции через регистры)

– удаление лишних команд

– снижение "стоимости" программы
(есть «дорогие» и «дешевые» команды с точки зрения времени их выполнения процессором)

– использование «машинных идиом» (например: xor ax,ax для обнуления регистра)

– слияние, дробление и развертывание циклов, иногда требующееся из-за технических особенностей аппаратуры
– учет векторных и конвейерных свойств архитектуры