Процессы идентификации и структура компилятора с языка АЛГОЛ-68

А. Н. Терехов

Рассматриваются некоторые языковые особенности АЛГОЛа68, повлиявшие на предлагаемую в статье структуру компилятора. Постулируются основные принципы, положенные в основу компилятора: последовательная обработка текста, отсутствие прямых количественных ограничений на число различных объектов в программе; описывается схема кодирования. Приводятся алгоритмы просмотров компилятора, в которых подробно описываются вопросы построения различных таблиц, участвующих в идентификации, их структура, а также работа с ними.

1. Во многих языках программирования используются различные символьные обозначения (идентификаторы, знаки действия и т.п.). При трансляции с этих языков возникает проблема распознавания таких обозначений, приписывания им некоторого смыслового значения (переменная, знак действия, строка для печати и т.д.), установления эквивалентности двух обозначений. Имеются два подхода к решению этой проблемы – контекстный и бесконтекстный.

При бесконтекстном анализе обозначений требуется, чтобы вся информация содержалась в самой записи обозначения. Обозначения различных классов представляются символами из непересекающихся множеств (обычно знаки действия не могут входить в состав идентификаторов; в языке ФОРТРАН идентификаторы целых переменных принято начинать с букв *i*, *j*, *k*, *l*, *m*, *n* и т.д.). Два обозначения считаются эквивалентными при их посимвольном совпадении.

При бесконтекстном анализе все вхождения обозначений делятся на два класса – описания и применения. В языке дается алгоритм, позволяющий каждому примененному вхождению обозначения найти его описание – определяющее его вхождение. Процесс установления связи между описаниями и применениями называется идентификацией. С ведением описаний повышается гибкость, выразительная сила языка, его надежность. Это связано с тем, что снимаются многие ограничения на запись обозначений; при необходимости нетрудно увеличить число различных классов обозначений с различным смысловым значением. Появляется возможность ввести некоторый контроль текста программы, например, если по ошибке программиста, перфоратора или устройства ввода исказится один из символов идентификатора, то эта ошибка может быть замечена еще во время трансляции (если искаженный идентификатор не совпадает с другим идентификатором того же типа, что маловероятно).

Идея идентификации получила дальнейшее развитие в языке АЛГОЛ68. Если в предыдущих языках программирования описывались идентификаторы только нескольких типов, то в АЛГОЛе 68 описываются идентификаторы бесконечного числа видов, идентификаторы вида, которыми можно пользоваться как обычными описателями, а также индикаторы операций, служащие для введения новых операций. Это, несомненно, создает новые удобства при программировании, но и усложняет структуру компилятора; в частности, процессы идентификации включаются почти во все части

"Программирование", №2, 1975, стр. 61-67

компилятора. В связи с этим дальнейшее обсуждение процессов идентификации будет проводиться в тесной связи со структурой компилятора с языка АЛГОЛ 68 на Промежуточный язык (п.я.), который с помощью специальных средств может быть реализован на вычислительных машинах. Основная особенность этого п.я. состоит в том, что возможен локальный перевод отдельных конструкций на машинный язык.

2. Укажем некоторые особенности, повлиявшие на структуру компилятора. В АЛГОЛе 68 есть конструкции, которые невозможно распознать без предварительной идентификации индикаторов. Например, (l : 3) и идентификатор b (если индикант **a** описан как бинарная операция), либо описанием переменной, в котором (l : 3) **a** является фактическим описателем (если **a** описан как индикант вида).

Таким образом, до распознавания конструкций, использующих индикаторы, нужно определить, какие индикаторы являются индикаторами вида, следовательно, произвести идентификацию индикаторов.

Все процессы идентификации в данном компиляторе состоят из двух фаз: построение таблицы определяющих вхождений и собственно идентификация. По определению, определяющим вхождением индикатора является его вхождение в описание приоритета (для бинарных индикаторов) или в описание вида (для индикаторов вида). Описание приоритета начинается символом **prio**, а описание вида – символом **mode**. Таким образом, эти описания легко выделяются в тексте программы.

Другая особенность АЛГОЛа 68 связана с использованием индикаторов в качестве операций. Для каждого бинарного индикатора, выступающего в качестве операции, необходимо найти два описания: описание приоритета для определения порядка действия и описание бинарной операции, определяющее ее программу. Для унарных индикаторов, выступающих в качестве операции, ищутся только описания унарных операций (для всех унарных индикаторов фиксируется максимальный приоритет). Таким образом, в процессе идентификации индикаторов нужно различать бинарные и унарные индикаторы. В данном компиляторе идентификация индикаторов совмещена с работой магазинного автомата, просматривающего текст слева направо, который распознает отдельные конструкции языка. Такой автомат не сможет отличить унарные индикаторы от бинарных в описаниях операций, записанных следующим образом:

op + = (**int** *a*, *b*) **int**: …
op – = (**int** *a*) **int**: …

Очевидно, что в момент обработки индикатора + автомат еще не может определить количество операндов в этой операции. Назовем такие вхождения индикаторов *сомнительными*. Для их идентификации понадобится некоторая специальная техника.

В пересмотренном сообщении об АЛГОЛе 68 определено новое контекстное условие, влияющее на идентификацию операций. Рассмотрим пример:

(**op**? = (**int** *a*) **int**: *1*;
(**op**? = (**ref int** *a*) **int**: *2*;
int *k*; ? *k*; ? *1*))

Согласно первоначальному варианту АЛГОЛа 68, применение операции ? к переменной и к константе идентифицирует различные описания. Авторы языка сочли, что запрещение подобных программ не уменьшит выразительной силы АЛГОЛа 68, улучшит ясность программ и ликвидирует источник многих ошибок программистов. Определяется это контекстное условие следующим образом.

Пусть для применения некоторой операции не нашлось описания в минимальном объемлющем блоке. Поиск описания можно продолжить только при том условии, что виды операндов этой операции "не связаны" с видами операндов других операций, описанных в этом блоке (два вида называются связанными, если существует вид, который в твердой позиции приводится к каждому из этих видов).

3. Работа компилятора состоит из последовательных просмотров текста программы; каждый просмотр осуществляет обработку последовательных порций текста (для разных просмотров величина порции может меняться). В пределах одного просмотра никаких возвратов по тексту не происходит. Некоторые задачи, не связанные с текстом программы (например, обработка таблиц), решаются между просмотрами; им отдается вся доступная память. Существенное влияние на распределение задач по просмотрам оказало соглашение, состоящее в том, что каждый просмотр строит не более двух таблиц переменных размеров (в частности, магазинов).

Текст, передаваемый от просмотра к просмотру, состоит из кодов и ссылок на таблицы, причем во всех просмотрах компилятора коды и ссылки имеют постоянную длину 2 байта. Код всегда имеет 0 в старшем разряде, а ссылка 1.

Коды распределяются на стандартные коды и марки. Стандартные коды – это коды понятий, начинающихся словом 'символ' в русском варианте АЛГОЛа 68. Марки – это специальные коды, расставляемые транслятором при разметке конструкций. Число марок фиксировано.

4. *Просмотр 1*. На вход поступает последовательность символов собственно программы; каждый символ АЛГОЛа 68 представлен одним или несколькими байтам. Производится первичная обработка текста; конкретные представления символов заменяются внутренними кодами; выбрасываются непрагматические комментарии; изображения заменяются внутренними представления. Строится таблица исходных представлений идентификаторов, указателей полей и индикаторов (таблица сверток).

В первом просмотре вхождения индикаторов и СЛОВ заменяются ссылками на таблицу исходных представлений[*)].

Определяющие индикатор вхождения выделяются в тексте программы и заносятся в таблицу. Эта таблица отражает блочную структуру программы. О каждом индикаторе в таблице содержится следующая информация:

1) ссылка на таблицу исходных представлений;

2) указание на сорт индикатора – вида или бинарный (унарные индикаторы в таблицу не попадают);

3) приоритет (для индикаторов приоритета) или код индикатора вида. В качестве кода индикатора вида берется порядковый номер этого индикатора среди всех вхождений индикаторов вида в таблицу.

[*)] Ссылки на исходное представление даются для так называемой "посмертной выдачи", т. е. для распечатки информации о непредвиденных случаях в работе программы в терминах исходной программы.

Здесь же проверяется контекстное условие единственности для индикаторов. Описания приоритета исключаются из текста программы.

Просмотр 2. На вход поступает последовательность записей постоянной длины (2 байта), причем каждая запись есть либо стандартный код или марка (0 в старшем разряде), либо ссылка на таблицу исходных представлений (1 в старшем разряде). Тип ссылки определяется маркой, стоящей непосредственно перед ссылкой:

a) после марки 'слово' – ссылка на исходное представление СЛОВА;

b) после марок 'изображение' – ссылка на исходное представление изображения;

c) после марки 'индикатор' – ссылка на исходное представление индикатора.

Построена таблица определяющих индикатор вхождений. Основным результатом этого и следующего просмотров является выделение и разметка синтаксических конструкций программы. Осуществляется это с помощью некоторого магазинного автомата, который последовательно выбирает из текста по одной записи. В тот момент, когда на вход автомата поступит ссылка, заменяющая индикатор, происходит обращение к процедуре идентификации, которая пытается найти в таблице определяющих индикатор вхождений строку, идентифицируемую данным вхождением индикатора.

Пусть такая строка нашлась. Обозначим ее через *S*. Если *S* соответствует индикатору вида, то в результирующий текст выводится марка 'индикатор вида' и код этого индикатора из *S*, и автомат переходит к обработке следующей записи. Если *S* соответствует бинарному индикатору, то автомат исследует, может ли данное вхождение индикатора быть вхождением бинарного индикатора, и при положительном решении этого вопроса в результирующий текст выводится марка 'индикатор приоритета *N*' (*N* берется из *S*) и ссылка на исходное представление этого индикатора, затем автомат переходит к обработке следующей записи; в противном случае данное вхождение индикатора считается вхождением унарного индикатора.

Если соответствующей строки в таблице не нашлось, то данное вхождение индикатора считается вхождением унарного индикатора.

Таким образом, все неописанные индикаторы временно включаются в класс унарных индикаторов. Ошибочность такого включения выявляется во время проверки конкретных условий.

Как уже было отмечено, существуют тексты, в которых автомат не может отличить унарные индикаторы от бинарных. Такие вхождения индикаторов мы условились называть сомнительными.

Не кодировать сомнительные вхождения индикаторов нельзя, так как тогда процедура идентификации индикаторов войдет как составная часть еще в один из просмотров компилятора, что нежелательно. В данной реализации принято другое решение: все сомнительные вхождения индикаторов кодируются как бинарные индикаторы (такие ситуации возникают только тогда, когда один и тот же символ используется и как унарный, и как бинарный индикатор), а в следующем просмотре после уточнения типов всех индикаторов неправильно закодированные унарные индикаторы будут возвращены к исходному виду, причем для этого не потребуется никакого поиска по таблице (см. описание предыдущего просмотра).

По окончании второго просмотра каждый бинарный индикатор в тексте уже имеет информацию о своем приоритете (снабжен маркой 'индикатор

приоритета *N*'), следовательно, таблица бинарных индикаторов не нужна. В таблице индикаторов вида после окончания идентификации индикаторов не нужна служебная информация, отражающая блочную структуру программы. Поэтому после второго просмотра из таблицы определяющих вхождений индикаторов вида и приоритета выписываются ячейки, относящиеся к индикаторам вида. Из описанной выше техники кодирования индикаторов вида следует, что адрес соответствующей ячейки в новой таблице является линейной функцией от кода индикатора, в связи с этим код индикатора можно не включать в таблицу.

Таким образом, строка таблицы индикаторов вида состоит из:

a) ссылки на исходное представление,

b) ссылки на таблицу описателей (заполнятеся на следующем просмотре).

Просмотр 3 (обратный). Текст, просматриваемый с конца, состоит из кодов и ссылок на таблицы. Окончания всех конструкций АЛГОЛа 68 помечены специальными марками.

С помощью магазина информация об окончании конструкций переносится на начало конструкций, уточняется информация о вхождениях унарных индикаторов (в сомнительных вхождениях унарных индикаторов марка 'индикатор приоритета *N*' заменяется маркой 'индикатор'). Текст программы преобразуется в рамочную запись, при которой как начало, так и конец каждой конструкции помечены специальными марками; кроме того формулы переводятся в прямую польскую запись. Строится таблица описателей, вхождения описателей в текст заменяются ссылками на эту таблицу.

По окончании просмотра 3 производится проверка контекстных условий на виды по таблице описателей.

Просмотр 4. На вход поступает последовательность кодов и ссылок. Различаются следующие типы ссылок:

a) после марки 'слово' — ссылка на исходное представление СЛОВА;

b) после марок 'изображение' — ссылка на исходное представление изображения;

c) после марки 'индикатор вида' — код индикатора вида (который одновременно может служить и ссылкой на таблицу индикаторов вида);

d) после марки 'индикатор приоритета *N*' — ссылка на исходное представление бинарного индикатора;

e) после марки 'индикатор' — ссылка на исходное представление унарного индикатора;

f) после марки 'описатель' — ссылка на таблицу описателей.

Задачей данного просмотра является построение таблицы определяющих вхождений идентификаторов и операций. Эта таблица должна быть компактна, легка в построении, удобна для идентификации по ней. Таблица должна отражать блочную структуру программы и содержать всю необходимую информацию об идентификаторах и операциях. Очевидно, что перечисленные требования к таблице во многом противоречивы. В АЛГОЛе 68 между описаниями одного блока можно вставлять операторы, в частности, блоки; определяющие вхождения могут находиться после примененных. Таким образом, описания, принадлежащие одному блоку, могут быть сильно разбросаны по тексту и, следовательно, в таблице, которая строится в соответствии с текстом, что усложняет использование этой таблицы для идентификации. Удовлетворить в какой-то степени всем перечисленным

требованиям удалось следующим образом: во время просмотра текста программы строится таблица, структура которой подобна структуре текста программы (легкость в построении!); после окончания просмотра информация об описаниях каждого блока собирается и записывается единым массивом в таблице, после чего идентификация существенно упрощается. Такую переработку таблицы можно осуществить достаточно эффективно (за один просмотр таблицы).

Таблица определяющих вхождений идентификаторов и операций состоит из служебных и информационных строк. Каждая информационная строка содержит: ссылку на исходное представление; ссылку на таблицу описателей; сорт информационной строки (соответствует идентификатору вида, идентификатору метки, унарной операции или бинарной операции).

Заметим, что для операций ссылка на таблицу описателей позволяет получить описатель той подпрограммы, которой обладает данная операция.

Во время работы просмотра служебные строки таблицы содержат следующую информацию:

1) служебная строка, соответствующая началу блока:
   a) марка 'служебная строка начала блока';
   b) ссылка на соответствующую служебную строку конца блока;
2) служебная строка, соответствующая концу блока:
   a) марка 'служебная строка конца блока';
   b) ссылка на ближайшую служебную строку, текстуально следующую за данной.

Для удобства идентификации необходимо в каждом блоке знать, с какого места в таблице искать определяющие вхождения этого блока, а также как продолжить поиск в объемлющем блоке, поэтому после окончания работы просмотра таблица преобразуется. Выписываются все информационные ячейки для внешнего блока без его подблоков и т. д. После такого преобразования в служебных строках таблицы содержится следующая информация:

1) служебная строка, соответствующая началу блока:
   a) марка 'служебная строка начала блока';
   b) ссылка на конец списка определяющих вхождений этого блока;
2) служебная строка, соответствующая концу блока:
   a) марка 'служебная строка конца блока';
   b) ссылка на конец списка определяющих вхождений объемлющего блока.

С целью повышения быстродействия алгоритма идентификации одновременно с описанным выше преобразованием информация об идентификаторах и операциях разносится в самостоятельные таблицы. Здесь же выполняется проверка контекстного условия единственности. Заметим, что проверка этого условия для операций требует установления связанности видов операндов в описаниях операций с одинаковым внешним представлением. Попутно информационные строки, соответствующие таким операциям, связываются в цепные списки.

Просмотр 5. Поскольку предыдущий просмотр не преобразовывал текста программ, входной язык не изменился. К началу работы данного просмотра размечены все конструкции, текст программы переведен в рамочную запись, построены таблицы определяющих вхождений идентификаторов и операций и таблица описателей.

Задачей данного и следующего просмотров является перевод текста программы на п. я. Для правильного перевода необходимо установление видов конструкций исходной программы и восстановление операций приведения. Именно: для каждого приводимого определяются исходный (не зависящий от контекста) и окончательный (определяемый контекстом) виды, а также род контекстной позиции. На основании этой информации специальная программа определяет последовательность применений.

В определении видов конструкций существенную роль играет идентификация идентификаторов и операций. По каждому примененному вхождению идентификатора или операции происходит обращение к процедуре идентификации, которая просматривает таблицу, построенную в предыдущем просмотре, и находит определяющее вхождение. Опишем работу процедуры идентификации для случая идентификации идентификаторов*.

Пусть два указателя *p* и *q* в начале работы просмотра 5 ссылаются на строку, непосредственно предшествующую первой строке таблицы определяющих вхождений идентификаторов. Во время работы просмотра происходит следующее: по марке 'начало блока' оба указателя сдвигаются на конец списка описаний блока, служебная строка начала которого находится в таблице по адресу *q* + 1; по марке 'конец блока' указатель *p* возвращается на конец списка описаний объемлющего блока, а к указателю *q* добавляется 1.

При идентификации идентификатора осуществляется поиск в таблице в сторону уменьшения адресов, начиная с адреса *p*, причем служебные строки начала блока пропускаются, а по служебным строкам конца блока осуществляется обход описаний соответствующих им блоков (обход параллельных блоков).

При идентификации операций нужны некоторые дополнительные действия, чтобы обеспечить выполнение описанного выше контекстного условия на идентификацию операций. Согласно этому условию каждое описание операции, которое находится в некотором блоке, "запрещает" использование операций, описанных во внешних по отношению к данному блоках, если их описания связаны с данным описанием операции (мы называем два описания операций *связанными*, если эти операции имеют одинаковые внешние представления и виды их соответствующих операндов связаны).

В каждую информационную строку таблицы определяющих вхождений операций добавляются два поля:

- в поле З (запрет) перед началом работы просмотра 5 записывается 0;
- в поле А (адрес) находится адрес ближайшего предшествующего вхождения в таблицу операции с таким же внешним представлением. Это поле заполняется после просмотра 4.

При входе в блок в поля З всех операций, связанных с операциями данного блока и не запрещенных внешними блоками, записывается уровень вложенности данного блока.

При идентификации строки с полем З > 0 не учитываются.

При выходе из блока снимаются запреты, установленные этим блоком.

Просмотр 6 (обратный). В этом просмотре размечаются начала рамочных конструкций п. я., а также выполняются некоторые действия, связанные с так

---

* Заметим, что идентификация индикаторов вида и приоритета в просмотре 2 осуществляется аналогичным образом.

называемой 'балансировкой видов'. Этим заканчивается перевод текста программы с языка АЛГОЛ 68 на п. я.