Титульный лист материалов по дисциплине

ДИСЦИПЛИНА *Теория формальных языков*

ИНСТИТУТ *Информационных технологий*

КАФЕДРА *Вычислительной техники*

ВИД УЧЕБНОГО МАТЕРИАЛА *Лекция*

ПРЕПОДАВАТЕЛЬ *Унгер Антон Юрьевич*

СЕМЕСТР *3 семестр*

2. ФОРМЫ ОПИСАНИЯ ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Подобного рода метаязыком являются формы Бэкуса-Наура [6], которые впервые были применены для описания языка ALGOL. Форма позволяет описать практически каждый из существующих языков и является стандартом де-факто описания грамматической структуры современных языков программирования.

После того, как мы познакомимся с теорией формальных языков и грамматик, будет дано формальное определение форм Бэкуса-Наура. Сейчас же важно понять основной применяемый в них подход. Каждое предложение формы описывает один *символ* языка. Под символом здесь понимается любая синтаксически допустимая конструкция. Например, выражение в языке C++ есть последовательность переменных, констант и функций, соединенных знаками операций, оканчивающаяся на символ «точка с запятой». Рассмотрим для примера, как с помощью формы Бэкуса-Наура описать русский язык:

```
<предложение> ::= <подлежащее> <сказуемое>
<подлежащее> ::= <существительное> | <местоимение>
<сказуемое> ::= <глагол> <наречие> | <глагол> <прилагательное>
<глагол> ::= бежит | летел | летели
<существительное> ::= человек | самолет | самолеты
<прилагательное> ::= красивый | высокий | быстрый
<местоимение> ::= он | она | мы
<наречие> ::= высоко | быстро | вчера
```

Здесь необходимо сделать следующие замечания. Во-первых, данная форма ни в коей мере не претендует на представление грамматического строя русского языка, тем более, что естественные языки не относятся к контекстно-свободным. Во-вторых, с помощью данного описания можно построить очень ограниченное количество предложений. Другими словами, данный язык описанный язык *конечен*.

Тем не менее, представленная форма показательна тем, что она дает формальный способ строить предложения. Начиная с символа `<предложение>` и последовательно заменяя символы в правой части каждого правила формы, можно построить все *синтаксически правильные* предложения.

Забегая вперед, заметим, что то, что стоит слева от символа `::=` называется нетерминалом, а слова, выделенные жирным – терминалами. Приведем

пример того, как, на каждом шаге заменяя один (самый левый) нетерминал, можно вывести предложение, состоящее из одних терминалов:

<предложение>
<подлежащее> <сказуемое>
<существительное> <сказуемое>
человек <сказуемое>
человек <глагол> <наречие>
человек бежит <наречие>
человек бежит быстро

Здесь важно остановиться на следующих моментах:

1. Мы пользуемся формой для генерации синтаксически корректных предложений. Говорят, что такие предложения допускаются данным языком. В процессе трансляции исходного текста программы решается обратная задача: текст пишется человеком, а транслятор на этапе анализа выносит решение о том, допускается ли данный текст языком программирования.

2. Форма может использоваться одновременно и для генерации предложений языка и для распознавания, является ли предложение синтаксически корректным. Во втором случае для исходного предложения выбирается одно из правил, для которого левая часть заменяется правой. Выбор нужного правила составляет *основную задачу синтаксического анализа*.

3. Можно видеть, что с помощью приведенной формы можно составить *синтаксически* корректные предложения, не имеющие смысла (например, самолет летели красивый). В данном случае, говорят о *семантической* ошибке.

Гораздо полезнее формы Бэкуса-Наура оказываются в случае языков программирования. Рассмотрим еще один пример:

1. <выражение> ::= <операнд> | <выражение> + <операнд> | <выражение> – <операнд>
2. <операнд> ::= <множитель> | <операнд> * <множитель> | <операнд> / <множитель>
3. <множитель> ::= <число> | (<выражение>)
4. <число> ::= <цифра> | <число> <цифра>
5. <цифра> ::= **0** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9**

Это пример арифметического выражения, допускающего такие операции, как сложение, вычитание, умножение и деление целых чисел. Терминалами в данном случае являются десятичный цифры.

Данная форма – пример бесконечного языка, о чем свидетельствуют правила 1 и 4, включающие *рекурсию* – нетерминал в левой части присутствует и в правой.

Форма допускает вложенные выражения, заключенные в круглые скобки (правило 3), и с его помощью можно сгенерировать выражения вида `2+3+4` или `2 + 3*(10-17)`. Сделаем несколько замечаний:

1. Выражения могут быть сколь угодно сложными, их длина принципиально не ограничена.

2. Рассмотрим пример того, как с помощью данной формы сгенерировать выражение `1+2*3`:

```
<выражение>
<выражение> + <операнд>
<операнд> + <операнд>
<множитель> + <операнд>
1 + <операнд>
1 + <операнд> * <множитель>
1 + <множитель> * <множитель>
1 + 2 * <множитель>
1 + 2 * 3
```

Здесь мы снова на каждом шаге заменяем самый левый нетерминал. Эта процедура называется *левым порождением*.

3. Из рассмотренного примера можно видеть, что данная форма Бэкуса-Наура учитывает *приоритет операций*. В самом деле, определение `<операнда>` (правило 2) предшествует определению `<множителя>` (правило 3), что соответствует тому, что приоритет умножения выше, чем сложения. Изменение приоритет достигается с помощью заключения выражения в круглые скобки (правило 3). К слову, заметим, что современные языки программирования включают множество операций, так что расстановка приоритетов при разборе выражения достигается иначе.

В таком простом виде формы Бэкуса-Наура практически не применяются. На практике их заменяют так называемыми Расширенными Формами Бэкуса-Наура (РБНФ), которые обладают большей выразительной мощью.

2.3.3. Существующие ограничения

Несмотря на то, что РБНФ повсеместно используются для описания допустимых языковых конструкций, им присущи определенные ограничения. Во-первых, они применимы только к так называемым *контекстно-свободным* языкам. Однако, разбирать программу без оглядки на контекст нельзя.

Например, использование переменной в выражении возможно только при условии, что эта переменная определена и ей присвоено значение. Говорят, что выражение на языке C `i = 2` корректно только в контексте, которому предшествует определение `int i;`. Учесть каждый такой *контекст* формы Бэкуса-Наура не позволяют, поэтому в процессе разбора требуются дополнительные *контекстно-зависимые* проверки.

Теория формальных языков, о которой будет идти речь ниже, утверждает, что один и тот же язык можно описать бесконечным числом различных, *эквивалентных* грамматических правил. Подобная «бесконечность» приводит различным реализация транслятора одного и того же языка, каждый из которых обладает достоинствами и недостатками. Кроме того, качество целевого кода может различаться.

Попытки учесть с помощью грамматических правил также и контекстные зависимости языка как правило ни к чему не приводят. Описание при этом получается столь сложным, что его детальный анализ становится практически невозможным. На практике применяется другой подход. Набор грамматических правил оставляют свободным от контекста, а контекстные зависимости учитывают с помощью набора дополнительных проверок. Данный двухэтапный анализ согласуется с лучшей практикой разработки трансляторов языков программирования высокого уровня.

2.3.4. Семантика языка

К сожалению, описать семантику языка значительно сложнее, чем синтаксис. Здесь нет единого формального подхода. Математически описать набор семантических правил возможно, однако это приводит к столь сложным определениям, что использовать их на практике невозможно.

Формальная семантика основывается на *доказательствах*, с помощью которых доказывается смысловая корректность и непротиворечивость программы. Существует несколько способов формально описать семантику языка программирования. К их числу относят:

1. **Операционная семантика.** При этом смысл выражений языка вкладывается в их описание. Например, можно определить некую абстрактную машину, в которой выражения будут проецировать на переходы между состояниями. Разрешенный переход соответствует семантически корректному выражению.

2. **Денотационная семантика** (от англ. *denotation* – обозначение). Здесь смысл вкладывается в *обозначения*, т.е. выражения обозначают их величины. Широко применяется в λ-исчислениях.

3. **Аксиоматическая семантика.** При этом подходе аксиомы используются для доказательства смысловой корректности программы так же, как в математике они используются для доказательства теорем.

Любая из данных теорий в значительной степени усложняет проектирование транслятора, делает его нетривиальной задачей. Для целей упрощения разработки транслятора часто применяют другой подход. Например, с помощью *атрибутной грамматики* можно дополнить контекстно-свободную грамматику дополнительными контекстно-зависимыми аспектами. При этом каждому символу грамматики ставится в соответствие некий атрибут. Атрибутом является просто некоторое, ассоциированное с символом значение, которое может быть прочитано или изменено в процессе анализа. Если рассматривать операционную семантику, как абстракцию вычислительного процесса, т.е. последовательности шагов, приводящих к определенному результату, то атрибутная грамматика является разновидности операционной семантики. В этом смысле, в частности, символ-переменная должен быть определен до его первого использования в вычислениях.

Можно сделать вывод о том, что строгое определение семантики языка есть трудная задача. На практике часто используют простое описание семантика на простом человеческом языке, т.е. на примерах описывают то и в каких ситуациях допустимо применять те или иные синтаксические конструкции. Таким образом, синтаксический анализ говорит о том, правильно ли записана та или иная конструкция, а разработчик, руководствуясь сводом семантических правил, решает, допустима ли она в том или оном контексте.

Разумеется, данный неформальный подход должен быть формально описан разработчиком транслятора для того, чтобы язык не допускал никаких двусмысленностей. По этой причине разработка современных языков программирования, даже имеющих долгую историю, не прекращается. В язык не добавляются новый конструкции, но и исправляются те, которые могут породить неоднозначность.

Далее везде в книге будет использован именно таком неформальный подход к определению семантики языка.