Формальная семантика языков программирования

Аннотация: в лекции излагаются основополагающие принципы, история развития, существующие подходы и выразительные возможности семантического представления формальных теорий и языков программирования.

Семантика языка программирования

Считается, что в настоящее время средства для определения синтаксиса языка достаточно удобны и развиты. Увы, но этого нельзя сказать о средствах формального представления семантики языка. Понятно, что для использования языка программирования следует описать каждую конструкцию языка в отдельности, а также ее применение в совокупности с другими конструкциями. В языке существует множество различных конструкций, точное определение которых необходимо как программисту, применяющему язык, так и разработчику компилятора для этого языка. Программисту эти знания позволяют прогнозировать вычисления, производимые операторами программы. Разработчику описания конструкций необходимы для создания правильной реализации компилятора.

Обычно семантика определяется с помощью обычного текста. Сначала при помощи какой-либо грамматики (скажем, BNF-грамматики) описывается синтаксис конструкции, а затем приводятся примеры и комментарии для разъяснения семантики. Очень часто смысл комментариев неоднозначен, поэтому различные читатели воспринимают его по-разному. Программист может получить ошибочное представление о работе создаваемой программы, а разработчик может сформировать ошибочную реализацию некоторой конструкции языка. Для исключения ошибок нужен инструмент для ясного, точного и лаконичного определения семантики всего языка.

Приемлемое решение в отношении семантики языка найти чрезвычайно трудно. Причина кроется в том, что содержание, смысл конструкций во много раз сложнее их формы. Был разработан целый ряд методов для формального определения семантики. Приведем описания наиболее популярных из них.

Синтезируемые атрибуты

Ранние попытки точно задать семантику делались путем добавления расширений к BNF-грамматике языка программирования. Дополнительную информацию о семантике можно было извлечь из дерева грамматического разбора. Идея, предложенная Дональдом Кнутом, заключалась в том, чтобы сопоставить каждому узлу дерева разбора программы некоторую функцию, определяющую семантику этого узла. Подобная информация сохранялась в так называемых атрибутах. Семантика конструкции может представляться некоторой величиной или набором величин, связанных с конструкцией. Например, семантика выражения 3 + 4 может быть целым значением 7, типом int или строкой + 3 4.

Атрибутные грамматики

Атрибутные грамматики можно рассматривать как обобщение аппарата синтезируемых атрибутов. Они позволяют осмысливать конструкции в зависимости от окружающего контекста. В этом случае значения атрибутов передаются как вверх, так и вниз по дереву разбора. Дело в том, что смысл узла на дереве разбора может зависеть не только от его поддерева, но и от информации из другой части дерева.

Предусматривается следующий порядок использования атрибутных грамматик для определения смысла конструкций:

1. Атрибуты — атрибуты добавляются к символам грамматики. Для каждого атрибута определяется, является ли он синтезируемым или наследуемым.
2. Семантические правила — семантические правила добавляются к правилам подстановки.
3. Если нетерминал N появляется в левой части правила подстановки, то добавляемые семантические правила определяют синтезируемые атрибуты для N.
4. Если нетерминал A появляется в правой части правила подстановки, то добавляемые семантические правила определяют наследуемые атрибуты для A.

Именно наследуемые атрибуты инициируют передачу информации вниз по дереву разбора или по горизонтали этого дерева (в этом случае говорят о передаче информации от «братских» узлов).

Операционная семантика

Операционное определение языка программирования описывает выполнение программы, составленной на данном языке, средствами виртуального компьютера. Виртуальный компьютер определяется как абстрактный автомат. Внутренние состояния этого автомата моделируют состояния вычислительного процесса при выполнении программы. Автомат транслирует исходный текст программы в набор формально определенных операций. Этот набор задает переходы автомата из исходного состояния в последовательность промежуточных состояний, изменяя значения переменных программы. Автомат завершает свою работу, переходя в не-

которое конечное состояние. Таким образом, здесь идет речь о достаточно прямой абстракции возможного использования языка программирования.

Итак, операционная семантика описывает смысл программы путем выполнения ее операторов на простой машине-автомате. Изменения, происходящие в состояни машины при выполнении данного оператора, определяют смысл этого оператора.

С одной стороны, машина-автомат способна воспринимать задания, представляемые лишь на простом языке моделирования. С другой стороны, современный компьютер может служить универсальным интерпретатором такого языка. Подобный интерпретатор может быть реализован программой, которая становится виртуальной машиной для языка моделирования.

Таким образом, в операционной семантике требуются:

- транслятор, преобразующий операторы исходного языка программирования L в команды низкоуровневого языка моделирования;
- виртуальная машина (ВМ) для языка моделирования.

Аксиоматическая семантика

Данный подход основывается на применении аппарата исчисления предикатов и теории доказательств. Семантику каждой конструкции языка определяют, как некий набор аксиом или правил вывода, используемый для вывода результатов выполнения этой конструкции. Чтобы понять смысл всей программы (то есть разобраться, что и как она делает), эти аксиомы и правила вывода следует применять так же, как при доказательстве обычных математических теорем. Аксиомы и правила вывода используют для оценки значений переменных после выполнения

каждого оператора программы. В итоге, когда программа выполнена, формируется доказательство того, что вычисленные результаты удовлетворяют заданным ограничениям на их значения относительно входных значений. Словом, доказывается, что выходные данные являются корректными результатами вычисления функций, обрабатывающих соответствующие входные данные. Примером описанного подхода считается метод аксиоматической семантики, созданный Тони Хоаром.

Итак, метод аксиоматической семантики служит для доказательства правильности программы. Доказательство правильности демонстрирует, что программа выполняет вычисления, описываемые ее спецификацией. При доказательстве каждый оператор окружается логическими выражениями (условиями), которые задают ограничения на программные переменные. Они используются для определения смысла оператора. Программа представляется в виде хоаровских троек.

Правило вывода для условного оператора

Данное правило позволяет получить семантически правильный условный оператор с булевым условием ветвления B, оператором S1, который надо поместить в then - ветвь, и оператором S2, который следует разместить в else -ветви.

У правильного условного оператора должна быть одна точка входа и одна точка выхода. Это означает, что постусловие у S1 и S2 должно быть одинаковое, например Q.

Предусловия операторов S1 и S2 описываются достаточно сложно. С одной стороны, у них есть общая часть P, которая задает стартовые ограничения на «арифметические» переменные. С другой стороны, оператор S1 должен запускаться при истинном значении булева условия B, а оператор S2 — при его ложном значении. Отсюда вывод: предусловия должны быть представлены как логические перемножения прямого (инверсного) булева условия B (not B) и утверждения P.

Правило вывода для оператора цикла WHILE

Для цикла WHILE количество итераций заранее неизвестно, поэтому применяется самостоятельное правило вывода, основанное на использовании инварианта.

Анализ правила:

1. Слабейшее предусловие должно иметь истинное значение до начала цикла и должно гарантировать истинность инварианта.
2. Постусловие должно иметь истинное значение после окончания цикла при истинном значении инварианта.
3. В ходе выполнения цикла истинность инварианта не должна зависеть от вычисления булева выражения, управляющего циклом, и от операторов тела цикла.
4. Инвариант должен иметь истинное значение до начала цикла, во время выполнения цикла и после завершения цикла. Отсюда и имя — инвариант.

Денотационная семантика

В этом подходе пытаются синтезировать определение функции, которую вычисляет каждая программа, написанная на языке программирования. Спецификация языка создается как иерархия определений функций, вычисляемых каждой отдельной программной конструкцией.

В программе любая операция представляет собой некоторую математическую функцию. Для композиции этих функций в более крупные фрагменты используются выражения и операторы. В свою очередь, линейные последовательности операторов и условные ветвления также могут быть представлены функциями, составленными из функций отдельных компонентов этих конструкций. Цикл легко описывается рекурсивной функцией, составленной из компонентов, входящих в его тело. В конечном счете образуется функциональная модель всей программы. Примерами такого подхода к заданию семантики являются метод денотационной семантики Скотта и Стрэчи, а также метод функциональной семантики Миллза.

Денотационная семантика — наиболее строгий метод для описания смысла программ. Он основан на теории рекурсивных функций.

Здесь для каждой конструкции языка задается:

1) математический объект;

2) семантическая функция, которая отображает экземпляры конструкции в экземпляры математического объекта.

Поскольку математические объекты определены точно, они представляют точный смысл синтаксических конструкций языка.

Чаще всего в качестве математического объекта используется множество положительных целых чисел N. Поэтому смысл языковых конструкций определяется значениями математических целых чисел.

Задать денотационную семантику языка — это значит определить семантические функции отображения для всех его понятий. Название «денотационная семантика» происходит от слова denote (обозначать), так как математический объект обозначает смысл соответствующей синтаксической единицы.