Концепции языков программирования

Описание синтаксиса и семантики

Темы

Введение

- Синтаксис — форма или структура выражений, предложений и программных единиц
- Семантика — значение выражений, предложений и программных единиц
- Синтаксис и семантика составляют определение языка
  - Кто пользуется определением языка
    - Другие разработчики языка
    - Те, кто реализует язык
    - Программисты (пользователи языка)

Общая задача описания синтаксиса

Терминология

- Предложение — это строка символов из некоторого алфавита
- Язык — это множество предложений
- Лексема — это мельчайшая синтаксическая единица языка (например, $\star$, sum, begin)
- Элементарная лексема — например, идентификатор

Формальное описание языков

- Распознаватели
  - Устройство распознавания считывает входную строку и решает, принадлежит ли она языку
  - Пример: часть компилятора, осуществляющая синтаксический анализ
- Генераторы
  - Устройство, порождающее предложения языка
  - Определить корректность конкретного предложения можно сравнив его со структурой генератора

Формальные методы описания синтаксиса

- Форма Бэкуса-Наура (BNF) и контекстно-свободные грамматики
  - Наиболее распространенный метод описания синтаксиса языков программирования
- Расширенная BNF
  - Более легкое чтение и написание по сравнению с BNF
- Грамматики и распознаватели

BNF и контекстно-свободные грамматики

- Контекстно-свободные грамматики
  - Разработаны Наумом Хомским в середине 1950-ых гг.
  - Генераторы языка, предназначенные для описания синтаксиса естественных языков
  - Определяют класс языков, называемых контекстно-свободными языками

Форма Бэкуса–Наура (BNF)

- Форма Бэкуса-Наура (1959)
  - Изобретена Джоном Бэкусом для описания языка Algol 58
  - BNF эквивалентна контекстно-свободным грамматикам
  - BNF является метаязыком, используемым для описания других языков
  - В BNF классы синтаксических структур представляют как абстракции: они ведут себя как синтаксические переменные (также называемые нетерминальными символами)

Основы BNF

- Нетерминальные символы: абстракции BNF
- Терминальные символы: лексемы
- Грамматика: набор правил

Пример (правила BNF)

< ident_list > → identifier | identifier, < ident_list >

< if_stmt > → **if** < logic_expr > **then** < stmt >

Правила BNF

- У правила есть левая и правая стороны; правило состоит из терминальных и нетерминальных символов
- Грамматика — это конечное непустое множество правил
- Абстракция (или нетерминальный символ) может иметь более, чем одну правую часть

$$\begin{array}{lcl} \texttt{<stmt>} & \rightarrow & \texttt{<single\_stmt>} \\ & | & \textbf{begin}\ \texttt{<stmt\_list>}\ \textbf{end} \end{array}$$

Описание списков

- Синтаксис списков описывают рекурсивно

$$\begin{array}{lcl} \texttt{<ident\_list>} & \rightarrow & \texttt{ident} \\ & | & \texttt{ident,<ident\_list>} \end{array}$$

- Порождение — это многократное применение правил: от начального символа к предложению (содержащему только терминальные символы)

Пример грамматики

$$
\begin{array}{lcl}
\texttt{program} & \rightarrow & \texttt{<stmts>} \\
\texttt{stmts} & \rightarrow & \texttt{<stmt>} \mid \texttt{<stmt>;<stmts>} \\
\texttt{stmt} & \rightarrow & \texttt{<var>=<expr>} \\
\texttt{var} & \rightarrow & \texttt{a} \mid \texttt{b} \mid \texttt{c} \mid \texttt{d} \\
\texttt{expr} & \rightarrow & \texttt{<term>} + \texttt{<term>} \mid \texttt{<term>} - \texttt{<term>} \\
\texttt{term} & \rightarrow & \texttt{<var>} \mid \texttt{const}
\end{array}
$$

Пример порождения

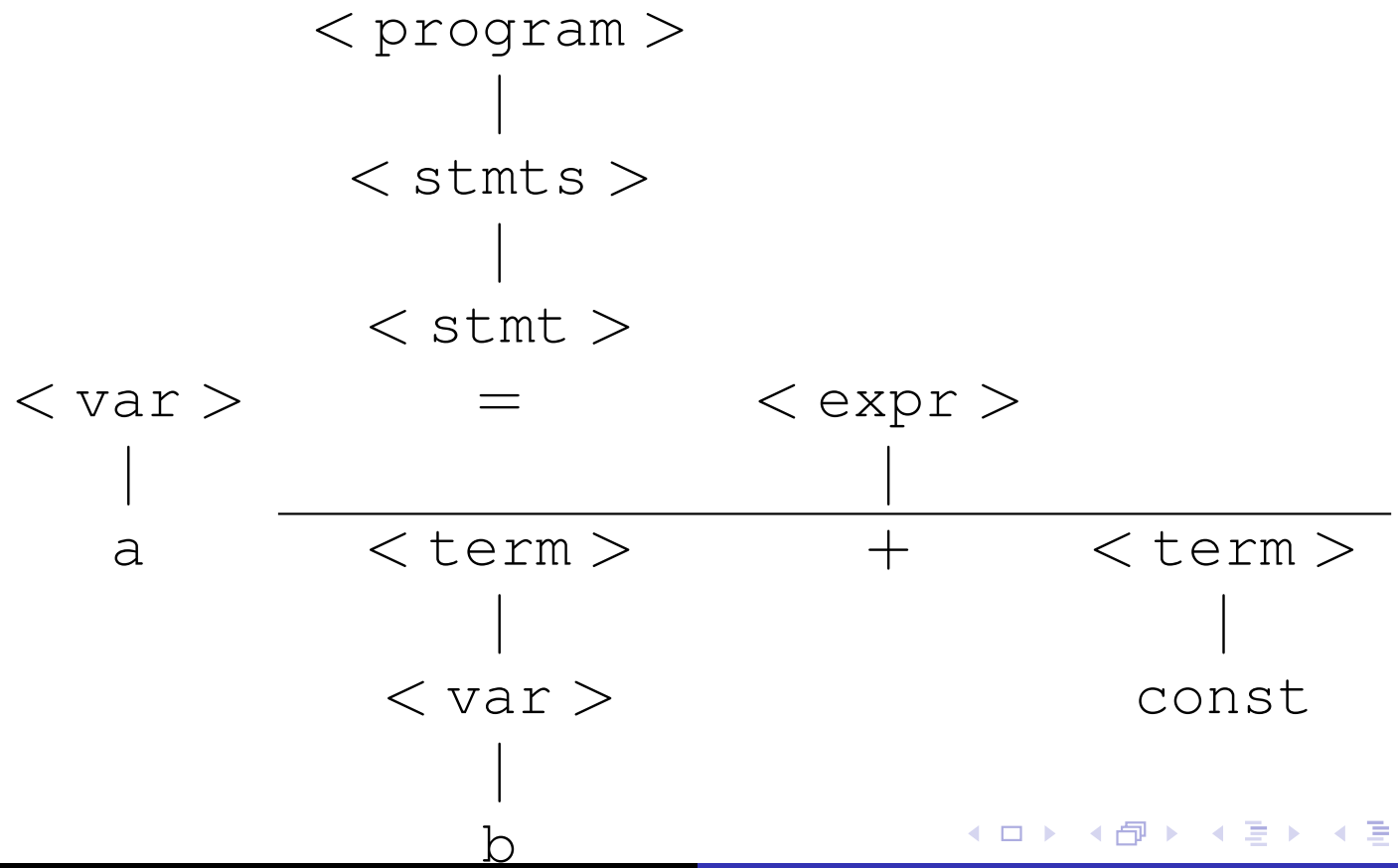
$$
\begin{aligned}
\texttt{program} &\Rightarrow \texttt{<stmts>} \\
&\Rightarrow \texttt{<stmt>} \\
&\Rightarrow \texttt{<var>=<expr>} \\
&\Rightarrow \texttt{a=<expr>} \\
&\Rightarrow \texttt{a=<term>+<term>} \\
&\Rightarrow \texttt{a=<var>+<term>} \\
&\Rightarrow \texttt{a=b+<term>} \\
&\Rightarrow \texttt{a=b+const}
\end{aligned}
$$

Порождение

- Каждая строка символов в порождении является сентенциональной формой
- Предложение — это сентенциональная форма, содержащая только терминальные символы
- «Левостороннее» порождение — такое, при котором всегда раскрывается самый левый нетерминальный символ
- Порождение не обязано быть ни левосторонним, ни правосторонним

Дерево парсинга

- Иерархическое представление порождения



Неоднозначные грамматики

- Грамматика неоднозначна, если она способна породить одну и ту же сентенциональную форму при помощи двух или более различных деревьев парсинга

Неоднозначная грамматика выражений

$$< \texttt{expr} > \rightarrow < \texttt{expr} > < \texttt{op} > < \texttt{expr} > | \texttt{const}$$
$$< \texttt{op} > \rightarrow / \mid -$$

Пример

Два дерева, соответствующие предложению
$\texttt{const} - \texttt{const}/\texttt{const}$

Однозначная грамматика выражений

- Использование деревьев парсинга для указания приоритета операторов позволяет избежать неоднозначности
  $< \texttt{expr} > \rightarrow < \texttt{expr} > - < \texttt{expr} > | \texttt{term}$
  $< \texttt{term} > \rightarrow < \texttt{term} > / \texttt{const} \mid \texttt{const}$
- Нарисуйте дерево парсинга для $\texttt{const} - \texttt{const}/\texttt{const}$

Ассоциативность операторов

- Ассоциативность операторов может быть задана в грамматике
  $<\texttt{expr}> \rightarrow <\texttt{expr}> + <\texttt{expr}> | \texttt{const}$
  (неоднозначно)
  $<\texttt{expr}> \rightarrow <\texttt{expr}> +\texttt{const} \mid \texttt{const}$ (однозначно)
- Дерево для $\texttt{const} + \texttt{const} + \texttt{const}$

Расширенная БНФ

- Опциональные части помещаются в квадратные скобки ([])

$$< \texttt{proc\_call} > \rightarrow \texttt{ident}[(< \texttt{expr\_list} >)]$$

- Альтернативные правые части помещаются в круглые скобки и разделяются вертикальными линиями

$$< \texttt{term} > \rightarrow < \texttt{term} > (+ \mid -)\texttt{const}$$

- Многократные вхождения (0 или более раз) заключаются в фигурные скобки ({})

$$< \texttt{ident} > \rightarrow < \texttt{letter} > \{\texttt{letter} \mid \texttt{digit}\}$$

БНФ и расширенная БНФ

- БНФ

$$\begin{aligned} &\langle \texttt{expr} \rangle \rightarrow \langle \texttt{expr} \rangle + \langle \texttt{term} \rangle \\ &\quad | \quad \langle \texttt{expr} \rangle - \langle \texttt{term} \rangle \\ &\quad | \quad \langle \texttt{term} \rangle \end{aligned}$$

- Расширенная БНФ

$$\begin{aligned} &\langle \texttt{expr} \rangle \rightarrow \langle \texttt{term} \rangle (+ \mid -) \langle \texttt{term} \rangle \\ &\langle \texttt{term} \rangle \rightarrow \langle \texttt{factor} \rangle \{(\star \mid /) \langle \texttt{factor} \rangle\} \end{aligned}$$

Атрибутивные грамматики

- Контекстно-свободные грамматики (КСГ) не способны полностью описать синтаксис языков программирования
- Расширения КСГ, добавляющие некоторую семантическую информацию к деревьям парсинга
- Основное назначение атрибутивных грамматик (АГ)
  - Описание статической семантики
  - Проектирование компиляторов (проверка статической семантики)

Атрибутивные грамматики

Определение

- Атрибутивная грамматика — контекстно-свободная грамматика $G = (S, N, T, P)$ со следующими дополнениями:
  - Для каждого символа грамматики x задано множество $A(x)$ атрибутов
  - Каждое правило содержит множество функций, определяющих атрибуты нетерминальных символов правила
  - Каждое правило содержит (возможно, пустое) множество предикатов, используемых для проверки согласованности атрибутов

Атрибутивные грамматики

Определение

- Пусть $X_0 \to X_1 \dots X_n$ — правило
- Функции вида $S(X_0) = f(A(X_1), \dots, A(X_n))$ определяют синтезируемые атрибуты
- Функции вида $I(X_j) = f(A(X_0), \dots, A(X_n))$ для $1 \leq j \leq n$ определяют наследуемые атрибуты
- Изначально, определены внутренние атрибуты на листовых узлах дерева

Атрибутивные грамматики

Определение

- Синтаксис
  $<\texttt{assign}> \rightarrow <\texttt{var}> = <\texttt{expr}>$
  $<\texttt{expr}> \rightarrow <\texttt{var}> + <\texttt{var}> \mid <\texttt{var}>$
  $<\texttt{var}> \rightarrow \texttt{A} \mid \texttt{B} \mid \texttt{C}$
- `actual_type`: синтезирован для элемента $<\texttt{var}>$ и $<\texttt{expr}>$
- `expected_type`: унаследован элементом $<\texttt{expr}>$

Атрибутивные грамматики

Продолжение

- Синтаксическое правило:
  $<\texttt{expr}> \rightarrow <\texttt{var}>[1] + <\texttt{var}>[2]$
  Семантические правила:
  $<\texttt{expr}>.\texttt{actual\_type} \leftarrow <\texttt{var}>[1].\texttt{actual\_type}$
  Предикаты:
  $<\texttt{var}>[1].\texttt{actual\_type} == <\texttt{var}>[2].\texttt{actual\_type}$
  $<\texttt{expr}>.\texttt{expected\_type} == <\texttt{expr}>.\texttt{actual\_type}$
- Синтаксическое правило: $<\texttt{var}> \rightarrow \texttt{id}$
  Семантическое правило:
  $<\texttt{var}>.\texttt{actual\_type} \leftarrow \texttt{lookup}(<\texttt{var}>.\texttt{string})$

Атрибутивные грамматики

Продолжение

- Как вычисляются значения атрибутов?
  - Если бы все атрибуты были наследуемыми, то значения можно было бы приписывать узлам дерева сверху вниз.
  - Если бы все атрибуты были синтезируемыми, то значения можно было бы приписывать узлам дерева снизу вверх.
  - Во многих случаях применяются оба типа атрибутов; поэтому значения нужно приписывать комбинированным способом (сверху вниз и снизу вверх)

Атрибутивные грамматики

Продолжение

$$< \texttt{expr} > .\texttt{expected\_type} \leftarrow \texttt{inherited from parent}$$

$$< \texttt{var} > [1].\texttt{actual\_type} \leftarrow \texttt{lookup(A)}$$
$$< \texttt{var} > [2].\texttt{actual\_type} \leftarrow \texttt{lookup(B)}$$
$$< \texttt{var} > [1].\texttt{actual\_type} =? < \texttt{var} > [2].\texttt{actual\_type}$$

$$< \texttt{expr} > .\texttt{actual\_type} \leftarrow < \texttt{var} > [1].\texttt{actual\_type}$$
$$< \texttt{expr} > .\texttt{actual\_type} =? < \texttt{expr} > .\texttt{expected\_type}$$

Семантика

- Нет единственной общепринятой нотации или единственного общепринятого формализма для описания семантики
- Операционная семантика
  - Описать значение программы, выполняя ее операторы на реальной машине или эмуляторе. Изменение состояния машины (памяти, регистров и т. д.) определяет значение оператора

Семантика

Продолжение

- Чтобы применить операционную семантику к языку высокого уровня, необходима виртуальная машина
- Чисто аппаратный интерпретатор был бы слишком дорог
- Чисто программный интерпретатор также имеет недостатки:
  - Подробные характеристики конкретного компьютера усложнят понимание действий
  - Подобное семантическое определение будет машинно-зависимым

Операционная семантика

- Предпочтительная альтернатива: полная эмуляция компьютера
- Процесс:
  - Построить транслятор (транслирует исходный код в машинный код идеализированного компьютера)
  - Построить эмулятор идеализированного компьютера
- Свойства операционной семантки
  - Подходит для неформального использования (руководства по языку и т. д.)
  - Очень сложна в формальном использовании (например, VDL), использовалась при описании семантики PL/I.

Семантика

- Аксиоматическая семантика:
  - Основана на формальной логике (исчисление предикатов)
  - Исходная цель: формальная верификация программ
  - Подход: Определить аксиомы или правила вывода для каждого типа оператора языка (чтобы разрешить трансформации выражений в другие выражения)
  - Выражения называются утверждениями

Аксиоматическая семантика

- Утверждение перед оператором — предусловие — определяет отношения между переменными и накладывает на них ограничения, которые должны выполняться в этой точке во время выполнения
- Утверждение, следующее за оператором — постусловие
- Самое слабое предусловие — наименее ограничительное предусловие, гарантирующее постусловие

Аксиоматическая семантика

- Пред–пост форма: $\{P\}$ оператор $\{Q\}$
- Пример: $a = b + 1\{a > 1\}$
  Возможное предусловие: $\{b > 10\}$
  Самое слабое предусловие: $\{b > 0\}$

Аксиоматическая семантика

Процесс доказательства корректности программы:

Желаемый результат — постусловие всей программы. Разбираем программу с конца до первого оператора. Если предусловие первого оператора соответствует спецификации программы, то программа корректна.

Аксиоматическая семантика

- Аксиома для операторов присвоения ($x = E$):

$$\{Q_{x \to E}\} x = E \{Q\}$$

- Правило следствия

$$\frac{\{P\} S \{Q\}, P' \Rightarrow P, Q \Rightarrow Q'}{\{P'\} S \{Q'\}}$$

Аксиоматическая семантика

- Правило вывода для последовательностей
  Для последовательности $S1; S2$:
  $\{P1\}S1\{P2\}$
  $\{P2\}S2\{P3\}$
  правило вывода таково:

$$\frac{\{P1\}S1\{P2\}, \{P2\}S2\{P3\}}{\{P1\}S1; S2\{P3\}}$$

Аксиоматическая семантика

- Правило вывода для циклов с предварительной проверкой логического условия
  Для цикла
  $\{P\}$*while B do S end*$\{Q\}$
  правило вывода таково:

$$\frac{(I \text{ and } B)S\{I\}}{\{I\}\text{while } B \text{ do } S\{I \text{ and}(\text{not } B)\}}$$

  где I — инвариант цикла (предположение индукции)

Аксиоматическая семантика

- Свойства инварианта цикла
  I должен удовлетворять следующим условиям:
  - $P \Rightarrow I$ (инвариант цикла должен выполняться вначале)
  - $\{I\}B\{I\}$ (оценка условия не должна повлиять на истинность инварианта)
  - $\{I \text{ and } B\}S\{I\}$ (I не меняется в результате выполнения тела цикла)
  - $(I \text{ and } (\text{not } B)) \Rightarrow Q$ (если I истинно и B ложно, то Q истинно)
  - Цикл останавливается (это может быть трудно доказать)

Аксиоматическая семантика

- Инвариант цикла I — это ослабленная версия постусловия цикла, также являющееся предусловием
- I должен быть достаточно слабым, чтобы выполняться перед циклом, но в комбинации с условием выхода из цикла, он должен быть достаточно сильным, чтобы обеспечить истинность постусловия

Семантика

Продолжение

- Оценка аксиоматической семантики:
  - Трудно разработать аксиомы или правила вывода для всех операторов языка
  - Хороший инструмент для доказательства корректности и отличная основа для рассуждения о программах, но не слишком полезна для пользователей языка и разработчиков компиляторов
  - Ограниченная пригодность в описании значения конструкций языка программирования для пользователей языка и разработчиков компиляторов

Семантика

Продолжение

- Денотационная семантика
  - Основана на теории рекурсивных функций
  - Наиболее абстрактный метод описания семантики
  - Первый вариант разработан Скоттом и Стрейчи (1970)

Денотационная семантика

- Процесс построения денотационной спецификации языка (не всегда легкий):
  - Определить математический объект для каждой сущности языка
  - Определить функцию, отображающую экземпляры сущностей языка на экземпляры математических объектов
- Значение конструкций языка определяются только значениями переменных программы

Семантика

Продолжение

- Различие между денотационной и операционной семантикой: В операционной семантике изменения состояний определяются закодированными алгоритмами; в денотационной семантике они определяются строгими математическими функциями

Денотационная семантика

- Состояние программы — это значения всех переменных

$$s = \{< i_1, v_1 >, < i_2, v_2 >, \ldots, < i_n, v_n >\}$$

- Пусть VARMAP — функция, которая, получая на вход имя переменной и состояние, возвращает текущее значение переменной

$$\mathrm{VARMAP}(i_j, s) = v_j$$

Семантика

Продолжение

- Десятичные числа
  - Следующее денотационное описание сопоставляет строковым представлениям десятичных чисел числовые значения

Семантика

Продолжение

$$< \texttt{dec\_num} > \rightarrow 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$$
$$\mid < \texttt{dec\_num} > (0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9)$$

$$\mathrm{M}_{\texttt{dec}}('0') = 0, \mathrm{M}_{\texttt{dec}}('1') = 1, \ldots, \mathrm{M}_{\texttt{dec}}('9') = 9$$
$$\mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >' 0') = 10 * \mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >)$$
$$\mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >' 1') = 10 * \mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >) + 1$$
$$\ldots$$
$$\mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >' 9') = 10 * \mathrm{M}_{\texttt{dec}}(< \texttt{dec\_num} >) + 9$$

Семантика

Продолжение

- Выражения
  - Отображает выражения на $Z \cup \{error\}$
  - Предполагаем, что выражения — это десятичные числа, переменные, или бинарные выражения с одним арифметическим оператором и двумя операндами, каждый из которых может быть выражением

Семантика

Продолжение

$$
\begin{aligned}
&\mathrm{M_e}(<\mathtt{expr}>, \mathtt{s})\Delta = \\
&\quad \mathtt{case} <\mathtt{expr}> \mathtt{of} \\
&\qquad <\mathtt{dec\_num}> \Rightarrow \mathrm{M_{dec}}(<\mathtt{dec\_num}>, \mathtt{s}) \\
&\qquad <\mathtt{var}> \Rightarrow \\
&\qquad\quad \mathtt{if}\ \mathtt{VARMAP}(<\mathtt{var}>, \mathtt{s}) == \mathtt{undef} \\
&\qquad\qquad \mathtt{then}\ \mathtt{error} \\
&\qquad\qquad \mathtt{else}\ \mathtt{VARMAP}(<\mathtt{var}>, \mathtt{s}) \\
&\qquad <\mathtt{binary\_expr}> \Rightarrow \\
&\qquad\quad \mathtt{if}(\mathrm{M_e}(<\mathtt{binary\_expr}>.<\mathtt{left\_expr}>, \mathtt{s}) == \mathtt{undef} \\
&\qquad\qquad \mathtt{OR}\ \mathrm{M_e}(<\mathtt{binary\_expr}>.<\mathtt{right\_expr}>, \mathtt{s}) = \\
&\qquad\qquad\qquad \mathtt{undef}) \\
&\qquad\qquad \mathtt{then}\ \mathtt{error} \\
&\qquad\qquad \mathtt{else}\ldots
\end{aligned}
$$

Семантика

Продолжение

- Операторы присвоения
  - Отображает множества состояний во множества состояний

$$
\begin{aligned}
&M_a(x := E, s)\Delta = \\
&\quad \texttt{if } M_e(E, s) == \texttt{error} \\
&\qquad \texttt{then error} \\
&\qquad \texttt{else } s' = \\
&\qquad \{< i_1, v_1 >, < i_2, v_2 >, \ldots, < i_n, v_n >\}, \\
&\qquad\quad \texttt{where for } j = 1, 2, \ldots, n, \\
&\qquad\qquad v'_j = \text{VARMAP}(i_j, s) \texttt{ if } i_j \neq x \\
&\qquad\qquad\quad = M_e(E, s) \texttt{ if } i_j == x
\end{aligned}
$$

Семантика

Продолжение

- Циклы с предварительной проверкой логического условия
  - Отбражает множества состояний во множества состояний

$$
\begin{aligned}
&M_l(\texttt{while B do L}, s)\Delta = \\
&\quad \texttt{if } M_b(\texttt{B}, s) == \texttt{undef} \\
&\qquad \texttt{then error} \\
&\qquad \texttt{else if } M_b(\texttt{B}, s) == \texttt{false} \\
&\qquad\quad \texttt{then s} \\
&\qquad\qquad \texttt{else if } M_{sl}(\texttt{L}, s) == \texttt{error} \\
&\qquad\qquad\quad \texttt{then error} \\
&\qquad\qquad\quad \texttt{else } M_l(\texttt{while B do L}, M_{sl}(\texttt{L}, s))
\end{aligned}
$$

Семантика

Продолжение

- Смысл цикла — это значения переменных программы после выполнения операторов в теле цикла предписанное число раз в предположении отсутствия ошибок
- По сути, цикл превращен из итеративной конструкции в рекурсивную; рекурсивный контроль определен математически при помощи других рекурсивных функций на состояниях
- Рекурсию легче описать строгим математическим образом, чем итерации

Семантика

Продолжение

- Оценка денотационной семантики
  - Может быть использована для доказательства корректности программ
  - Предоставляет точный метод рассуждения о программах
  - Может помочь при проектировании языка
  - Использовалась в системах порождения компиляторов
  - Из-за своей сложности не особенно полезна пользователю языка

Заключение

- БНФ и контекстно-свободные грамматики являются эквивалентными метаязыками
  - Хорошо пригодны для описания синтаксиса языков программирования
- Атрибутивная грамматика — формализм, пригодный для описания как синтаксиса так и семантики языка
- Три основных метода описания семантики
  - Операционный, аксиоматический, денотационный