

ДЕНОТАЦИОННАЯ СЕМАНТИКА ЯЗЫКОВ ПРОГРАММИРОВАНИЯ

Рассмотрим стандартную денотационную семантика языков программирования императивного типа.

Семантические определения

Вначале определим простейший язык программирования и зададим его семантику с помощью семантических функций, отображающих конструкции языка в некоторую семантическую область.

В дальнейшем будем расширять язык, вводя в него новые конструкции, добавляя новые семантические определения.

На начальном этапе не будем обращать внимания на детали изменения состояния. Кроме того, в языке не будет определяемых программистом имен.

СЕМАНТИЧЕСКИЕ ОПРЕДЕЛЕНИЯ

Прежде, чем определить язык программирования **L**, введем следующие синтаксические области:

C ∈ **Cmd** (команды)

E ∈ **Exp** (выражения)

c ∈ **Com** (примитивные команды)

P ∈ **Pre** (примитивные предикаты)

СЕМАНТИЧЕСКИЕ ОПРЕДЕЛЕНИЯ

Синтаксис языка **L** может быть задан следующим образом:

$$C ::= c \mid e \mid \underline{\text{if}} E \underline{\text{then}} C_0 \underline{\text{else}} C_1 \underline{\text{fi}} \mid C_0; C_1$$
$$E ::= p \mid \underline{\text{true}} \mid \underline{\text{false}} \mid \underline{\text{if}} E_0 \underline{\text{then}} E_1 \underline{\text{else}} E_2 \underline{\text{fi}}$$

Результатом выполнения команды является изменение состояния и в качестве подходящего значения команды будут выбираться функции изменения состояния.

СЕМАНТИЧЕСКИЕ ОПРЕДЕЛЕНИЯ

Таким образом, если S – множество состояний, то значением команды будет элемент из области $S \rightarrow S$.

Предположим, что выражение не имеет побочного эффекта, т.е. вычисление выражения не производит изменения состояния.

Ограничим также область значений выражений областью истинностных значений $T = \{\text{true}, \text{false}, \perp\}$.

Результат выражения зависит от состояния. Таким образом, значением выражения будет элемент из области $S \rightarrow T$.

СЕМАНТИЧЕСКИЕ ОПРЕДЕЛЕНИЯ

Из вышесказанного следует, что семантическими областями (областями значений) для языка L являются:

$t \in T$ (истинностные значения)

$\sigma \in S$ (машинные состояния)

$\gamma \in (S \rightarrow S)$ (значения команд)

$\omega \in (S \rightarrow T)$ (значения выражений)

УСЛОВНЫЙ ОПЕРАТОР И ПОНЯТИЕ СТРОГОЙ ФУНКЦИИ

Прежде, чем задать семантические функции, определим **условный оператор** и введем **понятие строгой функции**.

Определение 1. Для $x \in P$, где P – область, содержащая T как слагаемое ($P = P_1 + P_2 + \dots + P_n$), и $y, z \in D$.

$$x \rightarrow y, z = \begin{cases} y, & \text{если } (x|T) \equiv \text{true} \\ z, & \text{если } (x|T) \equiv \text{false} \\ \perp_D, & \text{если } (x|T) \equiv \perp_T \end{cases}$$

Определение 2. Функция $f: A \rightarrow B$ называется **строгой** (записывается **strict f**), если для $\perp \in A$ $f(\perp) = \perp \in B$, т.е.

$$\text{strict } f(x) = \begin{cases} \perp, & \text{если } x = \perp, \\ f(x), & \text{в противном случае.} \end{cases}$$

СЕМАНТИЧЕСКИЕ ФУНКЦИИ КОМАНД

Семантические функции команд $C: \text{Cmd} \rightarrow (S \rightarrow S)$ определяются следующим образом:

$C[[c]] = \text{strict } \gamma$, где γ – функция, ассоциируемая с командой "c";

$C[[e]] = I$ – тождественная функция (т.е. $I \equiv \lambda\sigma.\sigma$);

$C[[\text{if } E \text{ then } C_0 \text{ else } C_1 \text{ fi}]] = \lambda\sigma.E[[E]]\sigma \rightarrow C[[C_0]]\sigma, C[[C_1]]\sigma;$

$C[[C_0;C_1]] = C[[C_1]] \cdot C[[C_0]].$

Здесь и далее двойные скобки "[[" и "]" используются для выделения синтаксических конструкций, к которым применяется семантическая функция.

Семантическая функция E , определяющая значения выражений языка L , будет определена далее.

СЕМАНТИЧЕСКИЕ ФУНКЦИИ КОМАНД

Для лучшего понимания пункта, определяющего семантику **условной команды**, зададим состояние $\tilde{\sigma}$, которому применяется соответствующая функция.

Тогда:

$$C[\text{if } E \text{ then } C_0 \text{ else } C_1 \text{ fi}]\tilde{\sigma} = E[E]\tilde{\sigma} \rightarrow C[C_0]\tilde{\sigma}, C[C_1]\tilde{\sigma},$$

т.е. функция $E[E]$ применяется к состоянию $\tilde{\sigma}$ и результат используется для выбора альтернативы.

$C[C_i]\tilde{\sigma}$, $i = 0, 1$ означает применение соответствующей функции $C[C_i]$ к состоянию $\tilde{\sigma}$.

Запись вида $\gamma_1 \cdot \gamma_0$ в последнем пункте семантического определения означает функциональную композицию, т.е. $(\gamma_1 \cdot \gamma_0)\sigma = \gamma_1(\gamma_0\sigma)$.

СЕМАНТИЧЕСКИЕ ФУНКЦИИ ДЛЯ ВЫРАЖЕНИЙ ЯЗЫКА L

Семантические функции для выражений языка L имеют аналогичное функциональное определение.

$E : \text{Exp} \rightarrow (S \rightarrow T)$ и задаются следующим образом:

$E[\underline{p}] = \text{strict } \omega$, где ω – функция, ассоциированная с "p";

$E[\underline{\text{true}}] = \text{strict}(\lambda\sigma.\text{true})$;

$E[\underline{\text{false}}] = \text{strict}(\lambda\sigma.\text{false})$;

$E[\underline{\text{if } E_0 \text{ then } E_1 \text{ else } E_2 \text{ fi}}] = \lambda\sigma.E[E_0]\sigma \rightarrow E[E_1]\sigma, E[E_2]\sigma$;

Пусть например, σ – подходящее состояние, т.е. $\sigma \neq \perp$. Тогда $E[\underline{\text{true}}]\sigma = \text{true}$. Если же $\sigma \equiv \perp$, то $E[\underline{\text{true}}]\sigma = \perp$.

ТЕОРЕМА

Теорема 1. Значения всех команд и выражений языка L являются строгими функциями.

Доказательство. Тривиально, методом структурной индукции.

Смысл этой, теоремы в том, что если однажды что-то не завершилось при выполнении программы, то ничего уже сделать нельзя.

В дальнейшем при расширении языка L будем полагать, что **теорема 1** доказана для соответствующего расширения.

Отметим, что в приведенных определениях используются семантические области, образованные из базисных областей значений, таких как целостные, истинностные и т.д., являющиеся п.ч.п.

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

Прежде всего добавим в язык **L** команду вида:

while E do C od

Необходимо, чтобы эта команда означала то же самое, что и конструкция:

if E then C; while E do C od else e fi

Поэтому, если обозначить **$C[\underline{\text{while E do C od}}$** через **$\gamma'$** , то необходимо, чтобы

$\gamma' = \lambda\sigma. E[E]\sigma \rightarrow C[C; \underline{\text{while E do C od}}]\sigma, I\sigma = \lambda\sigma. E[E]\sigma \rightarrow (\gamma' \cdot C[C])\sigma, \sigma$

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

Рассматривая правую часть равенства как функцию H от γ такую, что $H(\gamma) = \lambda\sigma.E[E]\sigma \rightarrow (\gamma \cdot C[C])\sigma, \sigma$, т.е.

$H = \lambda\gamma.\lambda\sigma.E[E]\sigma \rightarrow (\gamma \cdot C[C])\sigma, \sigma$, необходимо показать, что $\gamma' = H(\gamma')$.

Таким образом, γ' должна быть **фиксированной точкой H** .

Кроме того, как было показано ранее, требуется также, чтобы это была **минимальная фиксированная точка H** .

Поэтому, требуется применение оператора минимальной фиксированной точки **fix**:

$$\gamma' = \text{fix } H$$

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

Итак, вводим дополнительно в язык L синтаксическую конструкцию

$C ::= \underline{\text{while}}\ E\ \underline{\text{do}}\ C\ \underline{\text{od}}$

и новый пункт определения семантической функции C :

$$C[\underline{\text{while}}\ E\ \underline{\text{do}}\ C\ \underline{\text{od}}] = \text{fix } (\lambda\gamma.\lambda\sigma.E[E]\sigma \rightarrow (\gamma \cdot C[C])\sigma, \sigma)$$

Часто требуется добавить новые черты в язык не для увеличения его семантической мощности, а просто для упрощения использования уже имеющихся в языке возможностей.

СПОСОБЫ ДОБАВЛЕНИЯ НОВЫХ СИНТАКСИЧЕСКИХ КОНСТРУКЦИЙ

Существуют два способа добавления новых синтаксических конструкций в язык.

Первый способ состоит в добавлении новых конструкций и соответствующих им семантических определений. Этот путь формально расширяет язык и усложняет доказательства, которые используют структурную индукцию. Этот способ использовался при добавлении конструкции **while E do C od** в язык **L**.

Альтернативным способом является сохранение ядра языка без изменений и введение дополнительных конструкций с помощью синтаксических равенств. Этот путь показывает, как программа может быть сведена к основным конструкциям ядра языка чисто синтаксическими средствами.

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

На практике используются оба способа, причем иногда оба сразу, давая определения обоих видов и доказывая их эквивалентность.

Рассмотрим в качестве примера использования **второго способа** добавление оператора **отрицания** " \neg ":

$$\neg E = \text{if } E \text{ then false else true fi}$$

Используя это равенство нетрудно доказать справедливость следующих утверждений:

$$\text{if } \neg E_0 \text{ then } E_1 \text{ else } E_2 \text{ fi} = \text{if } E_0 \text{ then } E_2 \text{ else } E_1 \text{ fi}$$

$$\neg(\neg E) = E$$

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

Докажем первое утверждение. Пусть $\omega = E[\underline{\text{if}} \neg E_0 \underline{\text{then}} E_1 \underline{\text{else}} E_2 \underline{\text{fi}}]$ и $\omega_r = E[E_r]$ для $r = 0, 1, 2$.

Требуется доказать, что $\omega = \lambda\sigma.\omega_0\sigma \rightarrow \omega_2\sigma, \omega_1\sigma$ (*)

Из семантического определения имеем:

$$\begin{aligned}\omega &= \lambda\sigma.E[\neg E_0]\sigma \rightarrow E[E_1]\sigma, E[E_2]\sigma = \\ &= \lambda\sigma.E[\neg E_0]\sigma \rightarrow \omega_1\sigma, \omega_2\sigma = \\ &= \lambda\sigma.E[\underline{\text{if}} E_0 \underline{\text{then}} \underline{\text{false}} \underline{\text{else}} \underline{\text{true}} \underline{\text{fi}}]\sigma \rightarrow \omega_1\sigma, \omega_2\sigma = \\ &= \lambda\sigma.(\omega_0\sigma \rightarrow \text{strict}(\lambda\sigma.\text{false})\sigma, \text{strict}(\lambda\sigma.\text{true})\sigma) \rightarrow \omega_1\sigma, \omega_2\sigma.\end{aligned}$$

Применив обе части доказываемого равенства (*) к некоторому произвольному σ . Если $\sigma = \perp$, то легко проверить, что равенство (*) имеет место. Если σ – подходящее состояние, то рассмотрим различные случаи, зависящие от значения $\omega_0\sigma$. Например, пусть $\omega_0\sigma = \text{true}$. Тогда:

$$\begin{aligned}\omega\sigma &= (\text{true} \rightarrow \text{strict}(\lambda\sigma.\text{false})\sigma, \text{strict}(\lambda\sigma.\text{true})\sigma) \rightarrow \omega_1\sigma, \omega_2\sigma = \\ &= \text{false} \rightarrow \omega_1\sigma, \omega_2\sigma = \omega_2\sigma = \omega_0\sigma \rightarrow \omega_2\sigma, \omega_1\sigma\end{aligned}$$

Аналогично доказываются и другие случаи, ч.т.д.

ЦИКЛИЧЕСКИЕ КОНСТРУКЦИИ И ИХ СЕМАНТИКА

Введем в язык L дополнительно команду **until E do C od**, определив ее с помощью следующего синтаксического равенства: **until E do C od = while E do C od**.

Доказательство того, что это определение эквивалентно альтернативному определению команды **until E do C od**:

$$C[\text{**until E do C od**}] = \text{fix } (\lambda\gamma.\lambda\sigma.E[[E]]\sigma \rightarrow \sigma, (\gamma \cdot C[[C]]\sigma)).$$

Расширим язык L еще двумя циклическими: конструкциями (**repeat C while E** и **repeat C until E**) с добавлением соответствующих семантических равенств:

$$C[\text{**repeat C while E**}] = \text{fix } (\lambda\gamma.(\lambda\sigma.E[[E]]\sigma \rightarrow \gamma\sigma, \sigma) \cdot C[[C]])$$

$$C[\text{**repeat C until E**}] = \text{fix } (\lambda\gamma.(\lambda\sigma.E[[E]]\sigma \rightarrow \sigma, \gamma\sigma) \cdot C[[C]])$$

ТЕОРЕМЫ ДЛЯ ОПРЕДЕЛЕНИЯ ДОПОЛНИТЕЛЬНЫХ КОНСТРУКЦИЙ

Следующие две теоремы могут служить как альтернативные определения дополнительных конструкций.

Теорема 2. repeat C while E = C; while E do C od

Теорема 3. repeat C until E = C; until E do C od

Сформулируем еще одну теорему.

Теорема 4. while E do C od = if E then repeat C while E else e fi

Для доказательства этих теорем, а также для доказательства других подобных утверждений используются методы, базирующиеся на понятии **фиксированных точек**.