Семантика основных конструкций языка C#

Комментарий к слайду

В данной лекции будут рассмотрены вопросы, относящиеся к понятийному аппарату, истории развития, существующим подходам и выразительным возможностям семантического представления формальных теорий и языков программирования. При этом существенное внимание будет уделено сопоставлению семантики языков объектно-ориентированного и функционального программирования. В качестве примеров языков программирования будут выступать уже знакомый нам по предыдущему курсу язык SML и изучаемый нами язык C#.

Содержание лекции

Комментарий к слайду

Коротко о содержании лекции.

В ходе лекции будут рассмотрены важнейшие научные исследования, относящиеся к эволюции подходов к математическому моделированию семантического представления формальных теорий и языков программирования. При этом будет предпринята попытка классификации существующих видов семантики.

Далее будет представлено неформальное введение в наиболее адекватный целям данного курса и достаточно широко распространенный на сегодня подход к семантике, а именно, так называемый денотационный подход. Теоретические рассуждения о семантике в формальных теориях (на примере теории вычислений Д.Скотта) будут проиллюстрированы представлением денотационной семантики подмножества языка программирования C#, ограниченного наиболее важными, основополагающими конструкциями.

При этом существенное внимание будет также уделено сопоставлению и сравнительному анализу синтаксического и семантического аспектов наиболее важных с точки зрения программирования классов конструкций языка программирования C#.

Другим аспектом сравнительного анализа будет сопоставление семантик подмножеств языков программирования SML и C# на основе общей теоретической формализации.

Лекция завершится обзором литературы для более глубокого исследования материала.

Важнейшие работы в области семантики (1)

1960-е – Х. Барендрегт (H. Barendregt) описал семантику лямбда-исчисления

1969 – Д. Скотт (Dana S. Scott) предложил использовать ***домены*** (особый вид множеств) для формализации ***денотационной*** семантики как функции вычисления значения синтаксически корректных конструкций языка

1979 – М. Гордон (Michael J.C. Gordon) исследовал денотационную семантику языков программирования

Комментарий к слайду

Кратко остановимся на наиболее значительных (с точки зрения данного курса) этапах эволюции теории и практики семантического анализа языков программирования.

В 60-х г.г. Х. Барендрегтом (H. Barendregt) была детально описана семантика лямбда-исчисления – математической формализации, поддерживающей языки функционального программирования.

Позднее, в конце 60-х г.г., Д. Скоттом (Dana S. Scott) было предложено использовать для формализации семантики математических теорий так называемые домены (пока будем неформально понимать их как особый вид множеств). При этом на основе доменов Д. Скоттом был предложен так называемый денотационный подход к семантике. Такой подход предполагает анализ синтаксически корректных конструкций языка (или, иначе, денотатов) с точки зрения возможности вычисления их значений посредством специализированных функций.

Далее, в 70-х г.г., М. Гордоном (Michael J.C. Gordon) был исследован аппарат денотационной семантики применительно к языкам функционального программирования и сделан вывод об адекватности и практической эффективности применения этого подхода для решения поставленной задачи.

Важнейшие работы в области семантики (2)

1964 – П. Лендин (Peter J. Landin) разработал семантику модели языка программирования в форме абстрактной машины на состояниях

1969 – Ч. Хоар (Charles A.R. Hoare) создал аксиоматический метод, моделирующий операторы языка программирования

1960-е – Р. Флойд (Robert W. Floyd) создал метод индуктивных утверждений для формализации семантики протекания информации в программе (графическая иллюстрация - блок-схемы)

Комментарий к слайду

Параллельным направлением изучения семантики был подход, исследовавший изменения, которые происходили в процессе работы программы на основе отслеживания смены состояний.

Одним из практических результатов работ в этом направлении стала разработка П. Лендином (Peter J. Landin) семантики модели языка программирования в форме абстрактной машины, существенно использовавшей понятие состояния.

Альтернативный подход к формализации семантики (который был осуществлен в рамках исследования так называемой операционной семантики языков программирования) привел к созданию Ч. Хоаром (Charles A.R. Hoare) аксиоматического метода, моделирующего отношения и причинно-следственные связи, возникающие между операторами языка программирования.

Развитие операционной семантики языков программирования привело Р. Флойда (Robert W. Floyd) к созданию так называемого метода индуктивных утверждений, который использовался для формализации семантики протекания информации в программе. При этом существенным преимуществом предложенного Р. Флойдом метода стала возможность интуитивно прозрачной и наглядной графической иллюстрации, основанной на блок-схемах, формализующих последовательность протекания информации.

Общие сведения о семантике (1)

Семантикой называется интерпретация (т.е. смысловое значение) абстрактного синтаксиса (т.е. множества допустимых конструкций языка), выраженное в терминах той или иной математически строгой модели.

Основные подходы к семантике:

1) ориентированные на компиляцию (семантика – множество преобразований над синтаксической моделью);
2) ориентированные на интерпретацию (семантика – множество описанных на метаязыке преобразований синтаксически правильных языковых конструкций).

Комментарий к слайду

После обсуждения обобщенных и концептуально важных требований к языкам программирования в целом, перейдем к неформальному обсуждению семантического подхода, способного обеспечить реализацию этих требований.

Прежде всего, расширим наивное представление о семантике языка программирования (или формальной теории) хотя и предварительным, но более конкретным определением этого понятия.

Семантикой будем называть интерпретацию (или, иначе, смысловое значение) абстрактного синтаксиса (а точнее, множества допустимых видов конструкций языка), представленное в терминах той или иной математически строгой формальной модели.

Как оказывается, все многообразие возможных подходов к семантике можно в основном представить всего двумя типами семантик, а именно, семантиками, ориентированными на компиляцию и семантиками, ориентированными на интерпретацию.

В дальнейшем под подходами к семантике, ориентированными на компиляцию, будем понимать такие подходы, в которых семантика представляет собой множество преобразований над синтаксической моделью в той или иной форме.

В отличие от предыдущего подхода, под подходами к семантике, ориентированными на интерпретацию, будем понимать такие подходы, в которых семантика представляет собой множество описанных на специально построенном метаязыке преобразований синтаксически правильных конструкций языка программирования.

Общие сведения о семантике (2)

Виды семантик, ориентированные на интерпретацию:

1) ***Операционные*** (смысл конструкций языка в терминах переходов абстрактной машины из одного состояния в другое), например, SECD-машина П. Лендина;
2) ***Пропозиционные*** (смысл конструкций языка в терминах множества формул, описывающих состояния объектов программы), например, подходы Хоара и Флойда;
3) ***Денотационные*** (смысл конструкций языка в терминах абстракции функций на состояниях программы), например, теория семантических доменов Д. Скотта.

Комментарий к слайду

Сразу заметим, что целям нашего курса в большей степени соответствует второй подход, как более универсальный в силу того обстоятельства, что в нем используется метатеория, т.е. формализация, моделирующая преобразования текста программ.

Выделим основные направления, существующие в рамках подхода к семантике, ориентированного на интерпретацию и свяжем их с рассмотренными нами в ходе лекции направлениями исследований.

Оказывается, что существуют три основных вида семантик, ориентированных на интерпретацию.

Во-первых, необходимо упомянуть об операционных семантиках. Значение конструкций языка в таких семантиках выражается в терминах переходов той или иной абстрактной машины из одного состояния в другое. В качестве показательных примеров абстрактных машин можно привести, в частности, так называемую SECD-машину П. Лендина, а также категориальную абстрактную машину.

Другим типом семантик, ориентированных на интерпретацию, являются так называемые пропозиционные семантики. В отличие от операционных семантик, значение конструкций языка в таких семантиках выражается в терминах множества формул, описывающих состояния объектов программы. В качестве примеров можно привести, в частности, аксиоматический метод Хоара и метод индуктивных утверждений Флойда.

Наконец, наиболее значимым для нас типом семантик, ориентированных на интерпретацию, являются денотационные семантики, в которых смысл конструкций языка представляется в терминах абстракции функций, оперирующих состояниями программы. В частности, данный подход иллюстрирует теория вычислений Д. Скотта, основанная на семантических доменах, которую и предлагаем вашему вниманию.

Теория вычислений Д. Скотта

Создана в 1969 г. для непротиворечивой формализации семантики языков на основе алгебры ***доменов*** – аналогов множеств, адекватно моделирующих самоприменимые функции и множества.

Последовательность построения теории:

1) перечисление ***стандартных*** (наиболее часто используемых) доменов;
2) определение ***конечных*** доменов с явно перечисляемыми элементами;
3) определение ***конструкторов*** (способов комбинирования) доменов.

Комментарий к слайду

Напомним, что теория вычислений Д. Скотта была создана до появления большинства современных языков программирования, а именно в конце 60-х г.г.

Существенно, что именно эта теория (в отличие от, скажем, классической логики и ряда других формальных систем) позволяет произвести адекватную (а именно, полную и непротиворечивую) формализацию семантики языков программирования.

Теория вычислений Д. Скотта основана на фундаментальном понятии домена, который будем неформально понимать как некоторый аналог множества, впрочем, в отличие от традиционных множеств, адекватно формализующий рекурсивно (т.е. на основе самоприменения) определенные функции и множества.

Сформулируем последовательность изложения теории вычислений Д. Скотта.

Для построения теории вычислений необходимо, во-первых, перечислить так называемые стандартные, или, точнее, наиболее часто используемые в рамках данной формализации, домены.

После перечисления стандартных доменов необходимо определить так называемые конечные домены, или, точнее, домены, элементы которых возможно перечислить явным образом.

Наконец, после перечисления доменов перейдем к определению конструкторов доменов, под которыми понимаются операции построения новых доменов на основе имеющихся, или, иначе, определим способы комбинирования доменов.

Конструкторы доменов

1) ***Функциональное пространство:*** $[D_1 \to D_2]=\{f|f: D_1 \to D_2\}$;
2) ***Декартово произведение*** – домен всевозможных n-ок: $[D_1 \times D_2 \times \ldots \times D_n]=\{(d_1 \times d_2 \times \ldots \times d_n)|d_1 \in D_1, d_2 \in D_2, d_n \ldots, \in D_n\}$;
3) ***Последовательность*** D*- домен всех конечных последовательностей вида $d=(d_1,d_2,\ldots,d_n)$ из элементов $d_1,d_2,\ldots,d_n,\ldots$ домена D, где $n>0$;
4) ***Дизъюнктная сумма*** $[D_1 + D_2 + \ldots + D_n]=\{(d_i,i)|\ d_i \in D_i, 0<i<n+1\}$, где принадлежность элементов d_i компонентам D_i однозначно устанавливается специальными функциями принадлежности.

Комментарий к слайду

Перечислив основные типы элементарных доменов, перейдем к их комбинированию посредством конструкторов.

Заметим, что, подобно ламбда-исчислению и комбинаторной логике, теория вычислений обладает весьма лаконичным набором способов комбинирования доменов. Как мы увидим далее, существуют всего четыре типа конструкторов. Тем не менее, такой набор является вполне достаточным для построения домена, моделирующего семантику сколь угодно сложной предметной области или языка программирования.

Приведем определения перечисленных способов комбинирования доменов.

Под функциональным пространством из домена D_1 в домен D_2 будем понимать домен $[D_1 \to D_2]$, содержащий всевозможные функции с областью определения из домена D_1 и областью значений из домена D_2: $[D_1 \to D_2] = \{f \mid f : D_1 \to D_2\}$.

Под декартовым (или, иначе, прямым) произведением доменов D_1, D_2, … D_n будем понимать домен всевозможных n-ок вида

$$[D_1 \times D_2 \times \ldots \times D_n]=\{(d_1 \times d_2 \times \ldots \times d_n) \mid d_1 \in D_1, d_2 \in D_2, d_n \ldots, \in D_n\}.$$

Под последовательностью D* будем понимать домен всевозможных конечных последовательностей вида $d=(d_1,d_2,\ldots,d_n)$ из элементов $d_1,d_2,\ldots,d_n,\ldots$ домена D, где $n>0$.

Наконец, под дизъюнктной суммой будем понимать домен с определением $[D_1+D_2+\ldots+D_n]=\{ (d_i, i) \mid d_i \in D_i,\ 0<i<n+1 \}$,

где принадлежность элементов d_i компонентам D_i однозначно устанавливается специальными функциями принадлежности.

Синтаксис подмножества языка C#

Формализуем синтаксис (а далее – и семантику) языка программирования $c_{\#}$ – весьма ограниченного подмножества C#:

```
E::=true|false|0|1|I|!E|E1==E2|E1+E2
```
(выражение)

```
C::=I=E|if(E)C1 else C2|while(E) C|C1;C2
```
(команда)

Комментарий к слайду

Поставим задачу формализации семантики языка объектно-ориентированного программирования C#. Заметим сразу, что в рамках данного курса будет рассматриваться не все множество возможных конструкций данного языка, а некоторое весьма ограниченное (хотя и вполне достаточное для иллюстрации основных идей курса) их подмножество, которое условно назовем языком программирования c# и будем именовать так в дальнейшем.

Прежде всего, рассмотрим синтаксис языка c#, т.е. перечислим основные типы конструкций, составляющих его.

Язык c# содержит множество выражений `E`, которые формализуются посредством БНФ в следующем виде:

```
E ::= true | false | 0 | 1 | I | !E | E1==E2 | E1+E2
```

Заметим, что выражения включают логические (`true` и `false`) и целочисленные (в ограниченном объеме: `0` и `1`) константы, множество идентификаторов (`I`), а также операции отрицания (`~E`), сравнения (`E1==E2`) и сложения (`E1+E2`).

Кроме того, язык c# содержит множество команд `C`, которые формализуются посредством БНФ в следующем виде:

```
C ::= I=E | if(E)C1 else C2 | while(E) C | C1;C2
```

Заметим, что команды включают присваивание (`I=E`), условие (`if(E)C1 else C2`), цикл с предусловием (`while(E) C`), а также последовательность команд (`C1;C2`).

Деление синтаксиса языка c# на выражения и команды во многом является условным и служит иллюстративным целям.

Семантика подмножества C# (1)

Порядок построения семантики – теории вычислений:

1) определение синтаксических доменов (Ide, Exp и Com);
2) определение вычислительной модели;
3) определение семантических функций E, C и т.д. для отображения синтаксических конструкций в семантические;
4) определение семантических предложений.

При выполнении c#-программы происходит изменение состояния, состоящего из памяти (m, memory), которая характеризует соответствие идентификаторов и значений (связывание) либо имеет значение unbound.

Комментарий к слайду

Как уже отмечалось, в качестве математической формализации, моделирующей семантику языков программирования (в частности, языка c#), будет использоваться теория вычислений Д. Скотта.

Приведем порядок построения формальной модели семантики языка программирования c# согласно ранее представленному формальному описанию синтаксиса языка в терминах БНФ.

Прежде всего, необходимо дать определение синтаксических доменов (т.е. доменов, характеризующих основные синтаксические категории) для идентификаторов (домен Ide), выражений (домен Exp) и команд (домен Com).

Далее, следует представить определение вычислительной модели на основе синтаксических доменов.

Затем, необходимо перейти к определению семантических функций (E для домена Exp, C для домена Com и т.д.), которые отображают синтаксические конструкции языка программирования в соответствующие им семантические представления.

Наконец, следует сформулировать определение семантических предложений в терминах смены состояний программы.

Заметим, что при выполнении программы (в частности, написанной на языке программирования c#) происходит изменение состояния, состоящего из памяти (m, memory), которая в простейшем случае характеризует соответствие идентификаторов и значений (то есть, по сути, связывание переменной со значением) либо имеет значение unbound (характеризующее отсутствие связи идентификатора со значением, т.е. аналогичное свободной переменной).

Семантика подмножества C# (2)

Синтаксис определяется синтаксическими доменами:

Ide = {I|I - идентификатор};

Com = {C|C- команда};

Exp = {E|E- выражение}.

Состояние:

Параметр	Домен	Соотношение
Состояние	State	(s) State = Memory
Память	Memory	(m) Memory=Ide→[Value+{unbound}]
Значение	Value	(v)Value=Int+Bool

Комментарий к слайду

В соответствии с намеченной схемой рассуждений, перейдем к описанию синтаксических доменов, которые в полной мере определяют синтаксис языка c#:

Ide = {I | I – идентификатор};
Com = {C | C – команда};
Exp = {E | E – выражение}.

Совокупность всех возможных идентификаторов языка c# организуем в домен Ide, команд – в домен Com, и, наконец, выражений – в домен Exp.

Далее, сформулируем вычислительную модель на основе состояний программы языка c#, для наглядности систематизировав ее в виде следующей таблицы:

Параметр	Домен	Соотношение
Состояние	State	(s) State = Memory
Память	Memory	(m) Memory = Ide → [Value + {unbound}]
Значение	Value	(v)Value = Int + Bool

Заметим, что состояние программы в произвольный момент времени определяется состоянием «памяти» абстрактной машины той или иной формы. При этом под памятью понимается отображение из домена идентификаторов в домен значений (т.е. аналог связывания переменной со значением в ламбда-исчислении). Для корректной обработки исключительных ситуаций, возникающих в случае свободных переменных, вводится дополнительный элемент unbound. Домен значений представляет собой дизъюнктную сумму доменов, содержащих существующие в языке c# типы Int и Bool.

Семантика подмножества C# (3)

Семантические предложения для денотатов выражений:

E : Exp → [State →[[Value × State] + {error}]] ;

E[E]s = (v,s'), если v - значение E в s, s'-состояние после означивания;

E[E]s = error, если возникает ошибка несоответствия типов.

Семантические предложения для денотатов команд:

C : Com → [State →[State + {error}]]

Комментарий к слайду

В соответствии с намеченной схемой рассуждений, перейдем к описанию семантических предложений, которые описывают значение денотатов (т.е. правильно построенных конструкций) языка c#.

Приведем семантические предложения для выражений языка программирования c#:

E : Exp → [State → [[Value × State] + {error}]];
E[E]s = (v,s'), если v – значение E в s, s'– состояние после означивания;
E[E]s = error, если возникает ошибка несоответствия типов.

Из приведенных соотношений следует, что вычисление значения выражения языка программирования c# приводит к такому изменению состояния, что происходит связывание переменной со значением, либо (в случае невозможности связывания по причине несоответствия типов переменной и значения) вырабатывается ошибка. При этом состояние программы изменяется с s на s'.

Приведем семантические предложения для команд языка программирования c#:

C : Com → [State → [State + {error}]] .

Из приведенных соотношений следует, что вычисление значения команды языка программирования c# приводит, вообще говоря, к изменению состояния, причем возможно возникновение ситуации (например, несоответствия типов в ходе присваивания), при которой вырабатывается ошибка.

Семантика подмножества C# (4)

Семантические предложения для денотатов выражений (начало, окончание см. след. слайд):

E [0]s = (0,s);

E [1]s = (1,s);

E [true]s = (true,s);

E [false]s = (false,s);

E [I]s = (m, I = unbound) error, → (m,I,s);

Комментарий к слайду

В соответствии с намеченной схемой рассуждений, перейдем к описанию семантических предложений, которые описывают значение конкретных денотатов (т.е. правильно построенных конструкций) языка c#. Рассмотрим семантические предложения для денотатов констант целочисленного типа языка c#:

E [0] s = (0, s);
E [1] s = (1, s);

Как видно из приведенных соотношений, денотатами констант целочисленного типа являются значения этих констант (в форме упорядоченных пар вида «значение»-«состояние»), причем смены состояния программы не происходит. Рассмотрим семантические предложения для денотатов констант логического типа языка c#:

E [true] s = (true, s);
E [false] s = (false, s);

Как видно из приведенных соотношений, денотатами констант логического типа являются значения этих констант (в форме упорядоченных пар вида «значение»-«состояние»), причем смены состояния программы не происходит. Рассмотрим семантическое предложение для денотатов идентификаторов языка c#:

E [I] s = (m, I = unbound) error, → (m, I, s).

Как видно из приведенного соотношения, при возможности связывания денотатами идентификаторов являются идентификаторы, связанные со значениями (в форме упорядоченных троек вида «значение в памяти»-«идентификатор»-«состояние»), причем смены состояния программы не происходит, а при невозможности – выдается сообщение об ошибке.

Семантика подмножества C# (5)

Семантические предложения для денотатов выражений (окончание):

$\underline{\mathbf{E}}[!E]s = (\underline{\mathbf{E}}[E]s=(v,s')) \rightarrow (\text{isBool } v \rightarrow (\text{not } v, s'), \text{error}, \text{error};$

$\underline{\mathbf{E}}[E_1==E_2]s = (\underline{\mathbf{E}}[E_1]s=(v_1,s_1)) \rightarrow (\underline{\mathbf{E}}[E_2]s_1=(v_2,s_2)) \rightarrow (v_1=v_2,s_2), \text{error}), \text{error};$

$\underline{\mathbf{E}}[E_1+E_2]s = (\underline{\mathbf{E}}[E_1]s=(v_1,s_1)) \rightarrow (\underline{\mathbf{E}}[E_2]s_1=(v_2,s_2)) \rightarrow (\text{IsNum } v_1 \text{ and IsNum } v_2 \rightarrow v_1+v_2,s_2), \text{error}), \text{error}), \text{error};$

Упражнение. Разработать семантические предложения для денотатов команд.

Комментарий к слайду

Рассмотрим семантические предложения для денотатов выражений языка c#:

$\underline{\mathbf{E}}\ [!E]\ s = (\underline{\mathbf{E}}\ [E]\ s=(v, s'))\ (\text{isBool} \rightarrow (\text{not } v, s'), \text{error}, \text{error};$

$\underline{\mathbf{E}}\ [E_1=E_2]\ s = (\underline{\mathbf{E}}\ [E_1]s = (v_1, s_1)) \rightarrow (\underline{\mathbf{E}}\ [E_2]\ s_1 = (v_2, s_2)) \rightarrow (v_1 = v_2, s_2), \text{error}), \text{error};$

$\underline{\mathbf{E}}\ [E_1+E_2]\ s = (\underline{\mathbf{E}}\ [E_1]\ s=(v_1, s_1)) \rightarrow (\underline{\mathbf{E}}\ [E_2]\ s_1 = (v_2, s_2)) \rightarrow (\text{IsNum } v_1 \text{ and IsNum } v_2 \rightarrow v_1 + v_2, s_2), \text{error}), \text{error}), \text{error}.$

Проанализируем полученные соотношения.

Денотатом отрицания выражения является отрицание его значения; причем состояние программы изменяется. В случае несоответствия типов или небулевости выражения генерируется сообщение об ошибке.

Денотатом присваивания является присвоенное значение в новом состоянии. В случае несоответствия типов генерируется сообщение об ошибке.

Денотатом сложения является значение суммы в новом состоянии. В случае несоответствия типов генерируется сообщение об ошибке.

В качестве упражнения предлагается самостоятельно разработать семантические предложения для денотатов команд языка программирования c#.

Кратко резюмируем итоги лекции.
В ходе лекции была представлена классификация подходов к семантике языков программирования, признан целесообразным денотационный подход, который проиллюстрирован на примере языка c# – ограниченного подмножества C#.

Сходство семантики подмножеств SML и C#

- Сходства семантики конструкций (команд и выражений) языков программирования SMaLl и c# прозрачны и очевидны
- Существует непосредственное отображение (типа «один к одному») конструкций рассмотренных языковых подмножеств друг в друга
- Денотационная семантика (формализованная посредством доменов на основе теории вычислений Д.Скотта) является адекватной моделью семантики выражений и команд рассмотренных языков программирования

Комментарий к слайду

Важнейшим выводом, к которому мы с неизбежностью приходим к завершению лекции, является заключение о том, что семантики рассмотренных нами языков программирования SMalL и c# (которые являются подмножествами реальных языков SML и C#) имеют весьма много общего.

Более того, сходства семантики рассмотренных нами конструкций (команд и выражений) языков программирования SMaLl и c# весьма прозрачны и очевидны. Для иллюстрации справедливости этого утверждения достаточно сопоставить результаты исследования семантики рассматриваемых языков программирования.

По результатам такого сопоставления можно придти к закономерному выводу о том, что существует явное, непосредственное отображение конструкций рассмотренных языковых подмножеств друг в друга, причем это отображение носит взаимно однозначный характер.

Попутно мы приходим к еще одному важному выводу. Формализация денотационной семантики на основе доменов с помощью теории вычислений Д.Скотта является адекватной моделью семантики выражений и команд рассмотренных языков программирования (а также функционального и объектно-ориентированного подходов в целом). Таким образом, семантика языков функционального и объектно-ориентированного программирования достаточно близка к семантике формальных теорий, на которых они основаны (в частности, это справедливо для ламбда-исчисления и языков SML и C#). Кроме того, теория вычислений является актуальной и адекватной формализацией семантики, а денотационный подход – наиболее целесообразным для моделирования семантики различных классов языков программирования.

Завершить изложение хотелось бы, указав на аналогию теории вычислений Д.Скотта и ламбда-исчисления (как метатеорий) со средой Microsoft .NET (как метасредой), которые принципиально обеспечивают возможность «погружения» в себя различных языков и подходов к программированию.

Библиография (1)

1. Barendregt H.P. The lambda calculus (revised edition), Studies in Logic, 103, North Holland, Amsterdam, 1984
2. Scott D.S. Domains for denotational semantics. ICALP 1982, 577-613
3. Gordon M.J.C. The denotational description of programming languages. Springer-Verlag, 1979.
4. Landin. P.J. The mechanical evaluation of expressions. Computer Journal, 6:308-320, January 1964.
5. Hoare C.A.R. An axiomatic basis for computer programming. CACM 12(10):576-580, 1969

Комментарий к слайду

К сожалению, в рамках времени, отведенных на одну лекцию, можно лишь в общих чертах охарактеризовать семантику языка программирования. Для более детального ознакомления с особенностями, достижениями и проблемами в области семантики рекомендуется следующий список литературы:

1. Barendregt H.P. The lambda calculus (revised edition), Studies in Logic, 103, North Holland, Amsterdam, 1984
2. Scott D.S. Domains for denotational semantics. ICALP 1982, 577-613
3. Gordon M.J.C. The denotational description of programming languages. Springer-Verlag, 1979.
4. Landin. P.J. The mechanical evaluation of expressions. Computer Journal, 6:308-320, January 1964.
5. Hoare C.A.R. An axiomatic basis for computer programming. CACM 12(10):576-580, 1969

Кратко остановимся на источниках. Работа [1] является энциклопедией ламбда-исчисления, основной формализации языков функционального программирования. В работах [2,3] рассматриваются вопросы, связанные с развитием денотационной семантики, и в частности, с ее представлением посредством доменов. Работа [4] содержит формализацию системы вычислений, основанной на состояниях. В работе [5] представлен вариант операторного подхода к семантике в форме аксиоматического метода.

Библиография (2)

1. Floyd R.W. A note on mathematical induction on phrase structure grammars. Information and Control 4(4): 353-358, 1961
2. Gunter C.A., Scott D.S. Semantic Domains. Handbook on theoretical computer science, Vol.B: Formal models and semantics (B):633-674, 1990
3. Stoy J.E. Denotational semantics: the Scott-Strachey approach to programming language theory. MIT Press, 1977

Комментарий к слайду

Продолжим обсуждение работ, посвященных исследованию семантики.

1. Floyd R.W. A note on mathematical induction on phrase structure grammars. Information and Control 4(4): 353-358, 1961

2. Gunter C.A., Scott D.S. Semantic Domains. Handbook on theoretical computer science, Vol.B: Formal models and semantics (B):633-674, 1990

3. Stoy J.E. Denotational semantics: the Scott-Strachey approach to programming language theory. MIT Press, 1977

В работе [1] излагается метод индуктивных утверждений, одно из математических оснований операционной семантики. В работах [2,3] рассматриваются вопросы, связанные с основами и развитием денотационной семантики, в частности, с ее формальными моделями и их приложением к реализации языков программирования.