Синтаксис и семантика. Абстрактные синтаксические деревья

Косарев Дмитрий

матмех СПбГУ

08 сентября 2022 г.

Дата сборки: 8 сентября 2022 г.

В этих слайдах

Посмотрим на понятия *синтаксиса* и *семантики*. Примеры языков и построение AST. Операционная семантика

Если кратко...

Синтаксис — это язык, на котором мы записываем что-либо

Семантика$_1$ — смысл, который имеем в виду, записывая слова на языке

Семантика$_2$ — смысл, конкретного предложения в языке

Интерпретация — сопоставление некоторому синтаксису некоторой семантики.

Пример. Синтаксис арифметических выражений

Неформальное описание: *арифметические выражения с 4мя бинарными операциями* $(+, *, -, /$ *и стандартными приоритетами) от целых чисел и переменной* x

Какие выражения являются или не являются арифметическими?

- $1 + 2 * 3$
- $(1 + 2) * (3 - 4))$
- $(-x)/2$
- $\dfrac{1 + 2 * x + x * x}{2}$
- $x^0 + 42$
- 3.1415
- $*$

Пример: Тип для языка арифметических выражений

Неформальное описание: *арифметические выражения с 4мя бинарными операциями* $(+, *, -, /$ *и стандартными приоритетами) от целых чисел и переменной* x.

```
type expr =
| Const of int
| VarCalledX
| Plus of expr * expr
| Asterisk of expr * expr
| Dash of expr * expr
| Slash of expr * expr
```

```
type op = Plus | Asterisk | Dash | Slash
type expr =
| Const of int
| VarCalledX
| BinOp of op * expr * expr
```

Это другой, также вполне допустимый вариант

Пример: формулы с целыми числами и кванторами

Неформальное описание:

- константы-числа и инфиксные бинарные операции $(+, /, *, -)$
- кванторы $\forall, \exists$ и переменные $(x, y, z, \dots)$
- (так называемые) *предикатные символы* для чисел $(>, <, \leqslant, \equiv, \neq)$
- логические связки: $\vee$ – или, $\wedge$ – и, $\Rightarrow$ – если ... то ..., $\neg$ – отрицание
- иногда добавляют логические константы: $\top$ (истина, true, top) и $\bot$ (ложь, false, bottom)

Примеры формул и не формул

- $\forall x\ \exists y\ (x > y)$
- $\exists x\ \forall y\ (x > y)$
- $\forall x\ \forall y\ \big((x \equiv y) \Leftrightarrow (y \equiv x)\big)$
- $\exists y\ (x > y)$
- $\exists x\ y\ (x + 1 > y)$
- $\exists 1\ (2 > 1)$

Пример: тип для формул

```
type var_name = char

type expr =
| Const of int
  (* 42 *)
| Var of var_name
  (* x *)
| Plus of expr * expr
  (* e1 + e2 *)
| Slash of expr * expr
  (* e1 / e2 *)
| ...
```

```
type phormula =
| Top
| Bottom
| ForAll of var_name * phormula
  (* ∀ x f *)
| Conj of phormula * phormula
  (* f /\ g *)
| Negation of phormula
  (* ¬f *)
| GreaterThen of expr * expr
  (* e1 > e2 *)
| ...
```

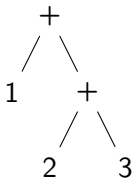
Абстрактные синтаксические деревья (AST)

Вспомним синтаксис арифметических выражений. Пример: $1 + (2 + 3)$

```
type expr =
| Const of int
| Plus of expr * expr
| ...
```

Это выражение будет выглядеть в AST как

```
Plus (Const 1,
      Plus (Const 2,
            Const 3))
```



Мы абстрагировались в том числе от:

- префиксности и инфиксности
- приоритетов и скобочек
- конкретных позиций в файле

Что может пойти не так с AST

- Может получиться AST не того языка
  - Рекомендуется в комментариях писать, что имелось в виду под тем или иным конструктором в AST

Что может пойти не так с AST

- Может получиться AST не того языка
  - Рекомендуется в комментариях писать, что имелось в виду под тем или иным конструктором в AST
- Надо избегать случаев, когда некоторым значениям AST сложно придать смысл. Пример

```
type expr = Plus of expr * expr | Const of int |  ...
let x = Plus (Const 1, Plus (Const 2, Const 3))

                (* vs *)

type expr = Plus of expr list | Const of int | ...
let y = Plus [ Const 1; Const 2; Const 3 ]
let z = Plus [  ]
```

Что может пойти не так с AST

- Может получиться AST не того языка
  - Рекомендуется в комментариях писать, что имелось в виду под тем или иным конструктором в AST
- Надо избегать случаев, когда некоторым значениям AST сложно придать смысл.
- Лучше избегать случаев, когда одну и ту же информацию можно представить двумя различными способами

```
type expr =
| Plus of expr * expr
| Op of op * expr list
| ...
```

Интерпретация

Интерпретация : Синтаксис $\longrightarrow$ Семантика

В домашнем задании будут подзадачи

- Задать синтаксис (AST) мини-языка
- Сделать интерпретатор (задать "адекватную" семантику)
- ...

Семантики бывают

1. Заданы в операционном стиле (*операционные семантики*)
   - назначение: смоделировать как что-то вычисляется
   - большого или малого шага
2. В денотационном стиле для некоторого домена $\mathcal{D}$ (*денотационные семантики*)
   - Объяснить смысл синтаксиса в уже известных терминах из $\mathcal{D}$

Операционная семантика для арифметических выражений (одна из)

```
type expr =
| Const of int
| Plus of expr * expr
| Slash of expr * expr
```

Операционная семантика для арифметических выражений (одна из)

```
type expr =
| Const of int
| Plus of expr * expr
| Slash of expr * expr
```

У нас в синтаксисе присутствуют целые числа, но нам никто не запрещает считать их вещественными...

Операционная семантика для арифметических выражений (одна из)

```
type expr =
| Const of int
| Plus of expr * expr
| Slash of expr * expr
```

У нас в синтаксисе присутствуют целые числа, но нам никто не запрещает считать их вещественными...

```
let rec interpret : expr → float = function
| Const n → float_of_int n
| Plus (l,r) → (interpret l) +. (interpret r)
| Slash (l,r) → (interpret l) /. (interpret r)
```

Возможное улучшение данной семантики: `interpret: expr → float option`

Малого и большого шага

Семантика **малого** шага

- каким-то образом демонстрирует промежуточные этапы

Семантика **большого** шага

- выдает сразу результат
- более эффективная (но менее познаваемая), чем малого

P.S.

Там ниже дополнительные примеры про написание типов

JSON

- Числа
- Строки
- Массивы
- Объекты как набор пар “ключ-значение”

JSON

- Числа
- Строки
- Массивы
- Объекты как набор пар “ключ-значение”

```
type json =
  | JNull
  | JBool of bool
  | JNum of float
  | JStr of string
  | JArray of json list
  | JObj of (string * json) list
```

Пример про почту (1/2)

```
type contact =
  { name : name
  ; emailContactInfo : email_contact_info
  ; postalContactInfo : postal_contact_info
  }
```

Пример про почту (1/2)

```
type contact =
  { name : name
  ; emailContactInfo : email_contact_info
  ; postalContactInfo : postal_contact_info
  }
```

Хочется, чтобы у контакта был *хотя бы один* адрес: либо электронной, либо физической почты.

Пример про почту (1/2)

```
type contact =
  { name : name
  ; emailContactInfo : email_contact_info
  ; postalContactInfo : postal_contact_info
  }
```

Хочется, чтобы у контакта был *хотя бы один* адрес: либо электронной, либо физической почты.

Что вы думаете о вот таком?

```
type contact =
  { name : name
  ; emailContactInfo :  email_contact_info option
  ; postalContactInfo :  postal_contact_info option
  }
```

Пример про почту (2/2)

```
type contact_info =
  | EmailOnly    of email_contact_info
  | PostOnly     of postal_contact_info
  | EmailAndPost of email_contact_info * postal_contact_info

type contact =
  { name        : name
  ; contactInfo : contact_info
  }
```

Если, посмотрев на тип, сразу понятно какие состояния корректные, а какие нет, то это считается хорошим дизайном.

Пример взят отсюда.

Содержательный пример (1/2) из [1]

```
type connection_state =
| Connecting
| Connected
| Disconnected

type connection_info =
  { state:                connection_state
  ; server:               inet_addr
  ; last_ping_time:       time option
  ; last_ping_id:         int option
  ; session_id:           string option
  ; when_initiated:       time option
  ; when_disconnected:    time option
  }
```

Содержательный пример (2/2)

Ссылки

OCaml for the Masses
Yaron Minsky
ссылка