Реализация сертифицированного интерпретатора для расширения просто типизированного лямбда-исчисления с концепт-параметрами

Юлия Белякова
julbel@sfedu.ru

4 апреля 2017
Ростов-на-Дону, Россия

PLC-2017
Языки программирования и компиляторы ’2017

О чём это?

1. Концепт-параметры. Формальная модель, описывающая простейшие «модуль-подобные» конструкции языков программирования — конструкции, обеспечивающие средства отделения интерфейса от реализации.
2. Front-end интерпретатора[1] для языка cpSTLC (STLC с концепт-параметрами). Отвечает за статическую проверку исходного АСД, в том числе, эффективную проверку определений модулей и типизацию термов.
3. Верификация интерпретатора. Доказательство соответствия интерпретатора формальной модели языка cpSTLC.

Формализм, front-end интерпретатора, и доказательства корректности реализованы в Coq и доступны на GitHub [1].

[1] Research in progress: вычисление термов ещё не реализовано.

Содержание

1. Концепт-параметры
2. Представление модулей в сертифицированном интерпретаторе
3. Заключение

Просто типизированное лямбда-исчисление
Simply Typed Lambda Calculus (STLC)

Типы

$$\tau ::= \texttt{Nat} \mid \texttt{Bool} \mid \tau \to \tau$$

Термы

$$e ::= x \mid \lambda x : \tau . e \mid e\ e \mid n \mid e + e \mid \texttt{if}\ e\ \texttt{then}\ e\ \texttt{else}\ e \mid \ldots$$

Пример терма:

$$(\lambda f : \texttt{Nat} \to \texttt{Bool} . \lambda x : \texttt{Nat}.\textbf{if}\ f\ x\ \textbf{then}\ x + x\ \textbf{else}\ 0)$$
$$(\lambda x : \texttt{Nat}.x > 42)\ 50$$

Результат вычисления: 100.

Пример программы с концепт-параметрами

```
concept MonoidNat                    (* interface *)
  ident : Nat
  op    : Nat -> Nat -> Nat
endc

model MNAdd of MonoidNat           (* implementation 1 *)
  ident = 0
  op    = \x:Nat.\y:Nat. x + y
endm

model MNMult of MonoidNat           (* implementation 2 *)
  ident = 1
  op    = \x:Nat.\y:Nat. x * y
endm

let accOrTwice = \m#MonoidNat.  (* m is a concept parameter *)
                 \x:Nat.\y:Nat. if x == m.ident
                                then m.op y y
                                else m.op x y
    (*    (sum of 3,5)       +      (square of 7)     *)
in  (accOrTwice # MNAdd 3 5) + (accOrTwice # MNMult 1 7)
```

Simply Typed Lambda Calculus with Concept Parameters

Типы

$$\tau := \mathtt{Nat} \mid \mathtt{Bool} \mid \tau \to \tau \mid \mathtt{C} \# \tau \qquad \textit{types}$$

$$\phi := \{f_i : \tau_i\} \qquad \textit{concept types}$$

$$\psi := (\mathtt{C}, \{f_i = e_i\}) \qquad \textit{model types}$$

Термы

$$e ::= x \mid \lambda x : \tau.\, e \mid e\ e \qquad \textit{STLC terms}$$

$$\mid n \mid e + e \mid \ldots \qquad \textit{nat/bool exprs}$$

$$\mid \lambda m \# \mathtt{C}.\, e \qquad \textit{concept abstraction}$$

$$\mid e \# \mathtt{M} \qquad \textit{model application}$$

$$\mid m.f \qquad \textit{function invocation}$$

Концепты и модели

Концепт описывает интерфейс модуля, то есть имена f_i и типы τ_i его элементов.
Модель задаёт реализацию указанного модуля, то есть ставит конкретные термы e_i в соответствие именам элементов f_i.

Требования к определению концептов и моделей

- Все имена в концепте должны быть различны.
- Типы элементов концепта должны быть корректны.
- Модель должна содержать те и только те имена, которые указаны в соответствующем концепте.
- Термы, реализующие элементы модели, должны быть корректны и иметь типы, указанные в соответствующих определениях концепта.

Представление модулей: проблема

Синтаксическое представление концептов/моделей

Список пар (<имя>, <тип>) / (<имя>, <терм>).

Представление модулей: проблема

Синтаксическое представление концептов/моделей

Список пар (<имя>, <тип>) / (<имя>, <терм>).

Семантическое представление концептов/моделей

?

Представление модулей: проблема

Синтаксическое представление концептов/моделей

Список пар (<имя>, <тип>) / (<имя>, <терм>).

Семантическое представление концептов/моделей

?

Для реализации формальной модели концепт-параметров в Coq можно использовать *список пар* [2—4]: это удобно для доказательства утверждений, но неэффективно.

Для реализации компилятора/интерпретатора [5] предпочтительней использовать эффективный *ассоциативный массив* (словарь, отображение из имён в типы/термы).

Что делать?

Представление модулей: решение

- Правильность определений концептов и моделей определяется в терминах списков пар.
- Интерпретатор проверяет определения концептов и моделей с помощью множеств и словарей.
- Все остальные части формальной модели и интерпретатора используют единое семантическое представление концептов и моделей на основе эффективных словарей (хранятся в таблице символов).
- Доказывается, что алгоритмы интерпретатора **корректны**: интерпретатор одобряет определения концептов и моделей тогда и только тогда, когда они правильны, проверка типов выполняется интерпретатором в полном соответствии с отношением типизации, и т.д.

Пример свойства корректности

Утверждение об уникальности имён [формальная модель]:

```
Definition ids_unique (fnames : list id) : Prop :=
  NoDup fnames.
```

Функция проверки уникальности имён [интерпретатор]:

```
Fixpoint ids_unique_b_rec (nmlist : list id) (nmset : id_set) : bool :=
  match nmlist with
  | nil        => true
  | nm :: nms => if ids_mem nm nmset then false
                 else ids_unique_b_rec nms (ids_add nm nmset)
  end.

Definition ids_unique_b (fnames : list id) : bool :=
  ids_unique_b_rec fnames ids_empty.
```

Свойство корректности

```
Lemma ids_unique_b__correct : forall (fnames : list id),
  ids_unique fnames <-> ids_unique_b fnames = true.
```

Заключение

Многие элементы определений, алгоритмов, и доказательств, связанных с концептами и моделями, могут быть повторно использованы для формализации и реализации модульных конструкций в компиляторах/интерпретаторах[2]:

- описание корректности определений модулей (уникальность имён, соответствие моделей концептам);
- алгоритмы проверки определений модулей;
- доказательства корректности алгоритмов по отношению к формальной модели.

[2] Coq-типы имён, типов, и термов объектного языка абстрагируются.