Синтез структур данных

Автор: Артем Мухин, 344 гр.
Руководитель: Дмитрий Мордвинов

Санкт-Петербургский государственный университет
Кафедра системного программирования

27 апреля 2018 г.

Синтез программ

Синтез программ — автоматический поиск программы, удовлетворяющей ограничениям, например:

- Формальной спецификации
- Примерам ввода-вывода
- Описанию на естественном языке
- Синтаксическому шаблону

Мотивация

- Структуры данных используются повсеместно
- Зачастую нужны структуры, не содержащиеся явно в стандартных библиотеках
  - Композиции нескольких стандартных структур
  - Стандартные структуры с особыми свойствами
- Программист часто ошибается и выбирает не самые оптимальные варианты
- Хочется, чтобы программист мог только описать требуемую структуру, а компьютер сам синтезировал оптимальную реализацию

Пример-1

- Нужна структура данных для хранения графа в виде рёбер и быстрого ответа на запрос "Какие рёбра инцидентны данной вершине?".

Спецификация:

```
record Edge { src: Int, dst: Int }
graph: Collection<Edge>
query findEdges(node: Int) =
    filter (λe. e.src = node or e.dst = node) graph
```

Пример-2

Возможные решения:

- Связный список. Выполнение запроса за O(n).
- Два `HashMap<Int, List<Edge>>`:
  - пары вида (src, все ребра из src)
  - пары вида (dst, все ребра в dst)

  . Выполнение запроса за O(1).

Синтезатор перебирает все возможные «решения» и выбирает из них оптимальное.

Пример-3

В итоге синтезатор генерирует такой код:

```
public class Graph {
    public static class Edge {
        public Edge(int src, int dst) { ... }
    }

    public Graph() { ... }
    public void add(Edge e) { ... }
    public void remove(Edge e) { ... }
    public Iterator<Record> findEdges(int node) { ... }
}
```

Существующее решение – Cozy (PLDI'16)

Cozy умеет решать только небольшое подмножество довольно простых задач (поиск элементов, удовлетворяющих предикату).
Cozy **не** умеет работать с:

- Более сложными комбинаторами: map, fold
- Арифметикой
- Нестандартными коллекциями

Постановка задачи

- Спроектировать язык спецификации
- Разработать алгоритм синтеза
- Разработать прототип ядра синтезатора

Структуры данных-1

```
* List<T>, Array<T>, Set<T>, Heap<T>
...
* Map<K, V>
Map.build          :: (t → (K, V)) → [t] → Map<K, V>
Map.get            :: K → Map<K, V> → V
Map.add            :: K → V → Map<K, V> → Map<K, V>
Map.size           :: Map<K, V> → Int

* FenwickTree<T>
FenwickTree.build :: (T→T') → Array<T> → FenwickTree<T>
FenwickTree.eval  :: Int → Int → T'
FenwickTree.get   :: Int → T
```

Структуры данных-2

```
Map.build f ks ∼ map (λk. (k, f k)) ks
Map.get key xs ∼ map snd (filter (λ(k, v). k = key) xs)
...
filter p (map f xs) = mapFilter (p . f) f xs
map f (filter p xs) = mapFilter p f xs
filter p' (mapFilter p f xs) = mapFilter (λx. p x  &  p' x) f xs
map f' (mapFilter p f xs) = mapFilter p (f' . f) xs
mapFilter p f (filter p' xs) = mapFilter (λx. p x  &  p' x) f xs
mapFilter p f (map f' xs) = mapFilter (p . f') (f . f') xs
```

- Синтезатор строит решение в терминах операций над коллекциями
- Для верификации решение сначала переписывается в терминах mapFilter-преобразований над списками

Пример преобразования

Решение в терминах операций над коллекциями:

```
let table: Map<Int, List<Edge>> = Map.build (λe.
    (e.src, filter (λe'. e'.src = e.src) graph)) graph
findEdges n = Map.get n table
```

Редукция к mapFilter:

```
findEdges n
~ map snd (filter (λ(src, _). src = n) (map (λe.
    (e.src, filter (λe'. e'.src = e.src) graph)) graph))
~ mapFilter (λe. e.src = n) (mapFilter (λe'.
    e'.src = e.src) id graph) graph
```

Перебор и верификация кандидатов

- Явный перебор и проверка с помощью SMT-решателей (подход Cozy и др.)
- **Сведение задачи перебора и верификации к SyGuS**

Syntax-Guided Synthesis (SyGuS)

SyGuS-синтезаторы строят программы, удовлетворяющие:

- синтаксической спецификации (грамматике)
- семантической спецификации (формуле логики первого порядка)

SyGuS Competition проводится каждый год начиная с 2014 г.

Построение грамматики для SyGuS

```
List<Edge>      ::= graph | Map<K, List<Edge>>.get
Map<K,V>.get ::= map snd (filter(λ(k,v).k=Obj<K>) Map<K,V>)
Map<K,V>        ::= Map<K,V>.build
Map<K,V>.build ::= map (λk. (k, Func<K,V> k)) List<K>
...
```

Специализируем и редуцируем грамматику:

```
map snd (filter (λ(k,v). k = Obj<K>) Map<K, List<Edge>>) ∼

map snd (filter (λ(k,v). k = Obj<K>) (map (λk.
    (k, Func<K,List<Edge>> k)) List<K>)) ∼

mapFilter (λ(k,v). k = Obj<K>) snd (map (λk.
    (k, Func<K,List<Edge>>)) List<K) ∼

mapFilter (λ(k,v). k = Obj<K>) Func<K,List<Edge>> List<K>
```

Построение ограничений для SyGuS

```
/* specification */
mapFilter (λe. e.src = n) id graph
/* candidate */
mapFilter (λ(k,v).k = Obj<Int>) Func<Int,List<Edge>> graph
```

```
mapFilter p f xs ∼ mapFilter p' f' xs
```

$$\forall x : p(x) \leftrightarrow p'(x) \;\;\&\;\; p(x) \rightarrow (f(x) = f'(x))$$

В итоге SyGuS приходит решению:

```
mapFilter (λe. e.src = n) (λe. filter (λe'.
    e'.src = e.src) graph) graph
```

Которое эквивалентно:

```
Map.get n (Map.build (λe. (e.src, filter (λe'.
    e'.src = e.src) graph) ) graph)
```

Реализация

Парсер языка спецификации реализован с помощью монадических парсер-комбинаторов (библиотека **parsec** языка Haskell)

- Парсер = примитивные парсеры + комбинаторы парсеров
- Преимущества: простота, гибкость, выразительность

Реализация

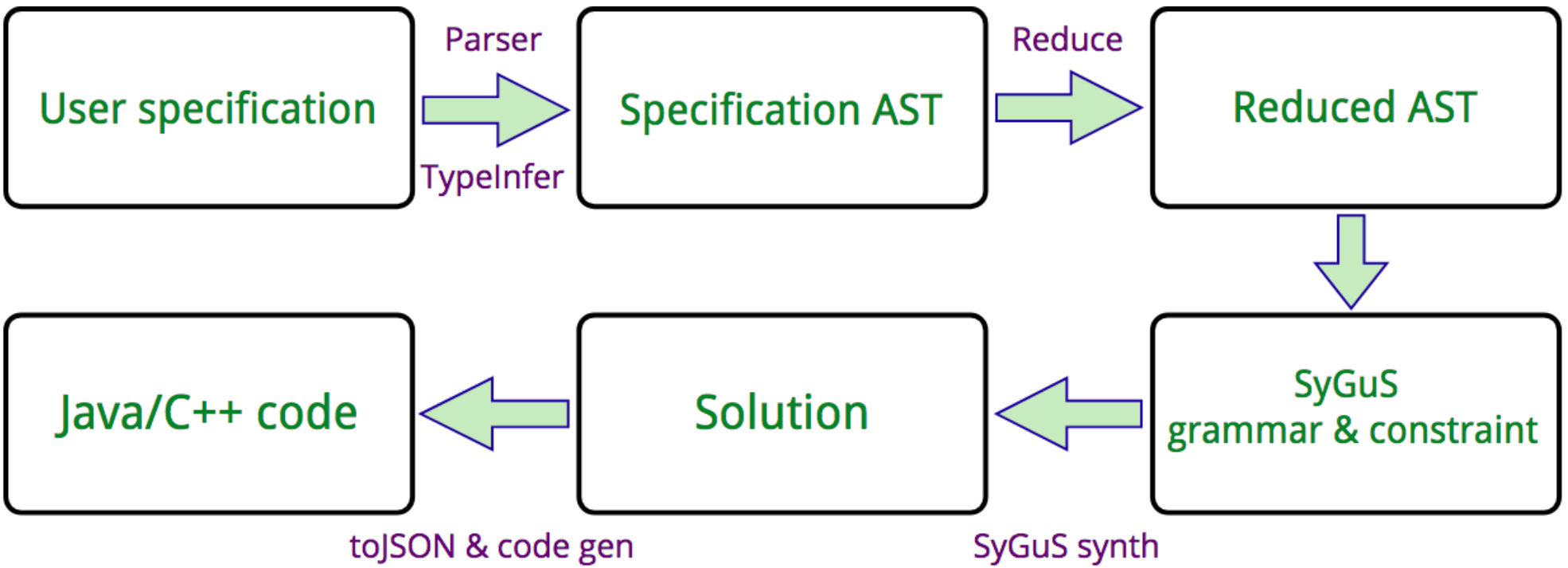
Для языка спецификации реализованы:

- Pretty-printer
- Редукция (map, filter $\rightarrow$ mapFilter)
- Вывод типов (основанный на алгоритме Хиндли-Милнера)
- Сведение к SyGuS
- Конвертация в JSON для последующей обработки генератором кода

Результаты

- Спроектирован декларативный язык спецификации структур данных
- Разработан алгоритм синтеза на основе редукции и сведения к SyGuS
- Разработан прототип ядра синтезатора, включающий в себя: парсер, вывод типов, редукцию, генерацию программ, конвертацию программ в JSON, сведение к SyGuS

Архитектура синтезатора



Планы на будущее

- Добавить поддержку fold
- Добавить больше структур данных и операций над ними
- Сравнить полученный синтезатор с существующими