Статическая проверка типов для Рефала

Выполнил: Романов А. С.

Научный руководитель: Коновалов А. В.

Постановка задачи

- РЕФАЛ динамически типизированный язык, где на вход приходит один аргумент и один аргумент возвращает
- Из-за своей динамической типизации в РЕФАЛ могут возникать ошибки с типом аргументов - это передача в функцию данных, не входящих в область определения функции — в таких случаях происходит аварийный останов программы.

Актуальность

- Уже существует неформальная нотация для типов данных
- Хотелось бы иметь инструмент для проверки программ на соответствие типам

```
<R5-Parse t.ErrorList s.Mode e.Tokens>
  == t.ErrorList t.Unit*

s.Mode ::= Classic | Extended

t.Unit ::= t.Function | t.Extern | t.SingleDeclaration | t.Include
  | t.NativeBlock
t.Extern ::= (Declaration t.Pos GN-Entry e.Name)
t.SingleDeclaration ::= (s.SingleDeclarationTag t.Pos s.ScopeClass e.Name)
s.SingleDeclarationTag ::= Enum | Swap
t.Include ::= (Include t.Pos e.Name)
t.NativeBlock ::= (NativeBlock t.Pos e.Code)
```

Формируем Типы

- Тип описывается как имя переменной, которой соответствует набор альтернатив.
  Пример: t.Tree ::= Leaf | (t.Tree t.ANY* t.Tree);
- Каждая альтернатива описывается как образцовое выражение, в котором, можно использовать знак «*» после терма как знак повторения
  Пример: e.Text ::= (s.CHAR*)*

Ограничения по Типам

- Выражения справа должны соответствовать типам переменных
- После e-переменных не может следовать «*»
- Не может быть более одной «*» на каждом скобочном уровне
- Тип может быть описан либо e-переменной целиком, либо цепочкой термов
- Все типы должны быть описаны кроме встроенных: s.NUMBER, s.COMPOUND, s.CHAR, s.ANY, t.ANY, e.ANY

Общий вид описания для Рефала

- **Определение типа:** Переменная “::=“ Альтернативы “;”
- **Альтернативы:** Простой тип {“|” Простой тип}
- **Простой тип:** e-переменная | {Терм} | {Терм} Терм”*” {Терм}
- **Терм:** s-переменная | t-переменная | “(” Простой тип ”)” | символ
- **Описание функции:** “<” Имя Простой тип “> == ” Простой тип ”;”

Алгоритм выполнения

- Сопоставление типа с образцом
- Подстановка типов переменных в правую часть
- Проверка, что фактический тип каждой функции аргумента вкладывается в формальный тип аргумента
- Проверка, что тип результатного выражения вкладывается в формальный тип результата функции

Сопоставление типа с образцом

- TE – типовое выражение
- TT – терм типа
- PE – образцовое выражение
- PT – образец уровня терм
- Если у нас сравнение TT TE ~ t.Var PE, то t.Var -> TT; TE ~ PE. Заметим, что если ранее данная переменная уже связывалась с неким типом, то новый тип должен совпадать со старым, иначе ошибка.
- Если у нас сравнение TE TT ~ PE t.Var, то t.Var -> TT; TE ~ PE.
- Если у нас сравнение (TE1) TE2 ~ (PE1) PE2, то TE1 ~ PE1; TE2 ~ PE2.
- Атом1 TE ~ Атом2 PE => TE ~ PE и Атом1 == Атом2, иначе ошибка.
- Если TT* ~ t.Var PE, то t.Var -> TT; TT* ~ PE.
- Если TE ~ e.Var, то e.Var -> TE.
- Если TT* ~ e.Var1 PE e.Var2, то e.Var1, e.Var2 -> TT*; TT* ~ PE

Вложение типа в тип

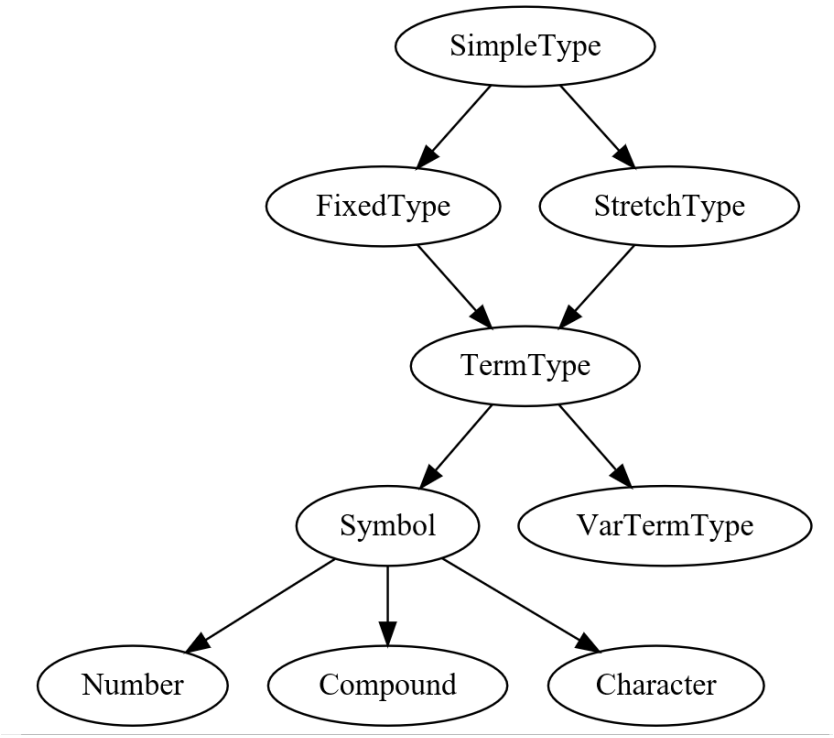
- t.Tree1 ::= Leaf | (t.Tree1 s.ANY t.Tree1);
- t.Tree2 ::= Leaf | (t.Tree2 t.ANY t.Tree2) | (t.Tree2 t.ANY t.Tree2 t.ANY t.Tree2)
- t.Tree1 ≤ t.Tree2
- [Leaf ≤ t.Tree2] && [(t.Tree1 t.ANY t.Tree1) ≤ t.Tree2]
- [[Leaf ≤ Leaf] || [Leaf ≤ (t.Tree2 t.ANY t.Tree2)] || [Leaf ≤ (t.Tree2 t.ANY t.Tree2 t.ANY t.Tree2)]] && [[(t.Tree1 s.ANY t.Tree1) ≤ Leaf] || [(t.Tree1 s.ANY t.Tree1) ≤ (t.Tree2 t.ANY t.Tree2)] || [(t.Tree1 s.ANY t.Tree1) ≤ (t.Tree2 t.ANY t.Tree2 t.ANY t.Tree2)]]
- [True || False || False] && [False || [t.Tree1 ≤ t.Tree2 && s.ANY ≤ t.ANY && t.Tree1 ≤ t.Tree2] || False]

Реализация

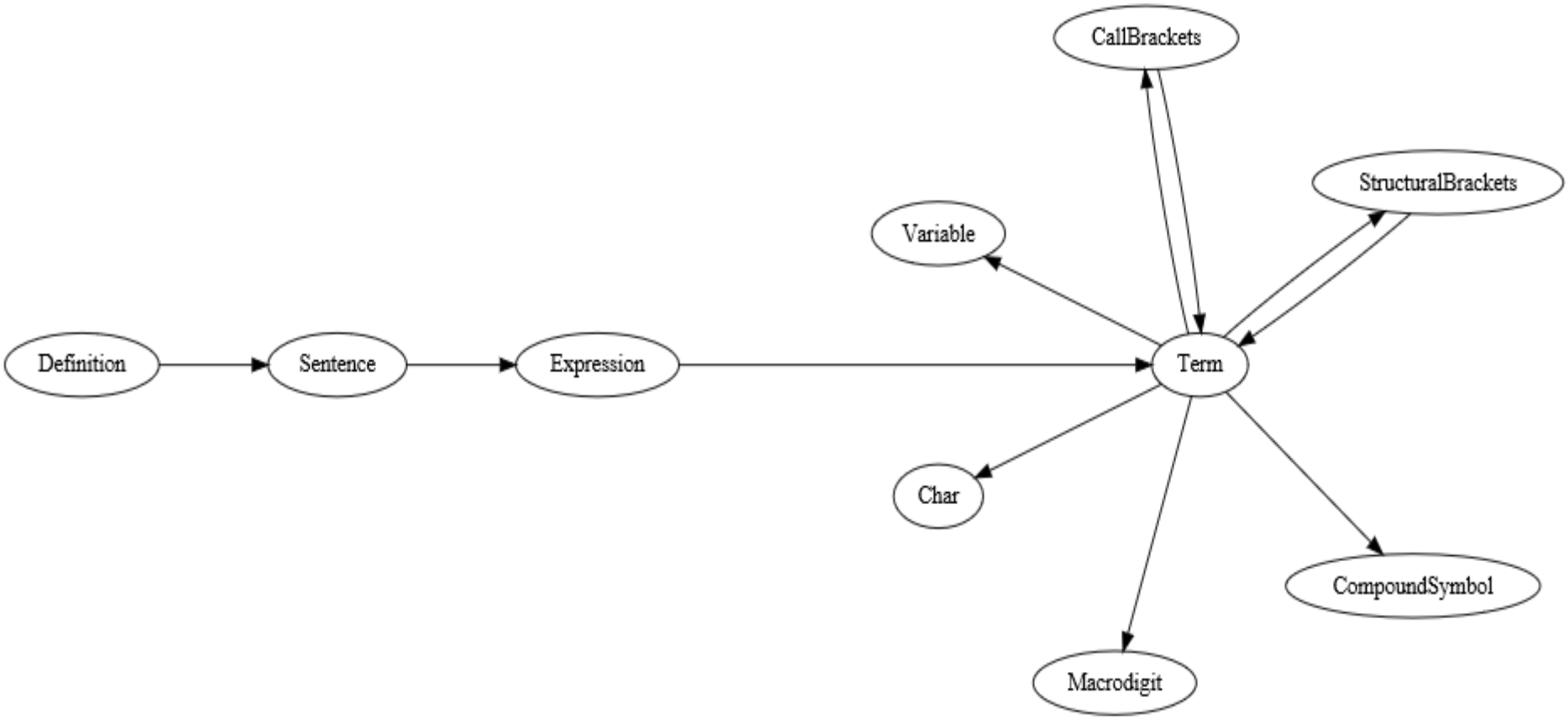
- Язык программирования: Java
- Фреймворк для автоматизации сборки: Maven
- Генератор лексических анализаторов: jflex
- Генератор синтаксических анализаторов: CUP

Дерево классов верификатора

```
enum Mode { S, T, E }
class Type {
  Mode mode;
  string Name;
  SimpleType[] constructors;
}
Class Func {
  string Name;
  SimpleType argument;
  SimpleType[] result;
}
```



Дерево классов Рефала



Тестирование

```
Sum {
  (S t.X) t.Y = <Sum t.X (S t.Y)>;
  NULL = t.Y;
}

Mul1 {
  NULL t.Y = NULL;
  (S t.X) t.Y = <Sum <Mul1 t.X> t.Y>;
}

$ENTRY Fact {
  NULL = (S NULL);
  (S t.Num) = <Mul1 <Fact t.Num> (S
t.Num)>;
}
```

```
t.Yum ::= s.ANY;
t.Num ::= NULL ;

<Sum t.Num t.Num> == t.Num;
<Mul1 t.Num t.Num> == t.Num;
<Fact t.Num> == t.Num;
```

Результат

Invalid number of arguments

Error, not match with description in Sum{

(S T.X) T.Y = <Sum T.X (S T.Y) > ;

NULL = T.Y ;

}

Invalid number of arguments

Can't match term with simpleType: <Mul1 T.X > expected [T.Num, T.Num]

Error, not match with description in Mul1{

NULL T.Y = NULL ;

(S T.X) T.Y = <Sum <Mul1 T.X > T.Y > ;

}

Спасибо за внимание