Semântica Formal - Trabalho Final

Alexandre Santos da Silva Jr. (00193093) Elias Saldanha (00194809)
Rodrigo Holztrattner (00218319)

2017/1

1 Resumo

Para o trabalho final da disciplina foi implementado um Interpretador da linguagem L1 seguindo as regras da semântica operacional no estilo big step. A linguagem de programação escolhida pelo grupo foi F, utilizando o ambiente de desenvolvimento do Visual Studio. O grupo obteve sucesso em desenvolver o interpretador com todas as características descritas e necessárias, incluindo o avaliador de expressões big step e inferência de tipos, para todas as funções da linguagem L1. O software também foi testado durante e ao final do desenvolvimento e apresentou resultados condizentes com os esperados nas condições criadas.

2 Implementação

Como citado anteriormente a implementação foi feita na linguagem F. O ambiente Visual Studio já era conhecido e utilizado comumente pelos membros do grupo, o que facilitou o setup inicial para desenvolvimento. Para divisão da implementação entre os integrantes, o projeto foi distribuído e versionado por meio da ferramenta github, permitindo melhor organização e contribuição no desenvolvimento do software.

3 Metodologia

O desenvolvimento do software foi feito em cascata por cada função, após os tipos iniciais serem designados com o auxílio da definição formal, cada função foi criada e testada separadamente, e ao longo do processo, com mais funções já desenvolvidas e funcionais por si, os testes começaram a incluir mais e mais funcionalidades para verificar a interação destas e sua integridade em conjunto, possibilitando novas iterações para correção e modificação caso necessário. O código é composto de diversas funções auxiliares para efeito de clareza, e duas funções principais: Eval e Typecheck.

Pela linguagem utilizada ser diferente do padrão utilizado na disciplina, OCaml, alguns desafios foram encontrados no aprendizado do F, mas a documentação e o próprio ambiente - que identifica automaticamente ainda antes da compilação potenciais problemas - foram extremamente úteis e possibilitaram um avanço rápido nesta etapa.

3.1 Eval

O eval realiza a avaliação da expressão fornecida tentando fazer o casamento da mesma com uma das expressões possíveis e por consequência acaba "abrindo" essa expressão maior em menores, resolvendo caso a caso realizando todas as operações necessárias. Tivemos que criar diversas funções auxiliares como a que procura o valor de uma variável em um ambiente, ou outra que verifica se uma expressão é um valor... Todas essas construções podem ser vistas melhor no código.

Todas as expressões de pattern matching foram feitas com base na especificação fornecida e por consequência nossa função eval é capaz de avaliar qualquer expressão descrita corretamente na linguamgem L1.

3.2 Typecheck

Já a função Typecheck é responsável pela inferência e validação de dada expressão quanto ao sistema de tipos da linguagem. A função foi desenvolvida em paralelo com Eval, compartilhando do uso de pattern matching para classificação das expressões, mas retornando um tipo como saída. Isso envolveu uma questão de design relevante: como informar, ao fim da execução da função, tanto se a expressão está de acordo com o sistema de tipos (tipicamente um valor booleano), como também o tipo para o qual a expressão resolve?

A solução tomada foi a introdução de um tipo especial, o "TyUnmatched", que essencialmente significa que a expressão avaliada é mal-formada quanto ao sistema de tipos. Neste caso, a avaliação da expressão quanto à semântica operacional nunca é realizada, e o erro é informado ao usuário por meio do console.

4 Resultados

O interpretador implementado inclui todas as funcionalidades presentes na linguagem L1 com o avaliador em Big Step. Ao compilar, o software não apresenta erros ou warnings e os testes realizados foram todos bem sucedidos, tanto na avaliação das expressões quanto no sistema de tipos. No tópico seguinte são apresentados alguns dos testes realizados para confirmar a validez da implementação.

5 Definições de Estruturas

```
type Variable = string

type Operator =
    | Sum
    | Diff
    | Mult
    | Div
    | Eq
    | Neq
    | Ls
    | Lse
    | Gr
    | Gre
    | And
    | Or

type Tipo =
    | TyInt
    | TyBool
    | TyFn of Tipo * Tipo
    | TyUnmatched
and
    Typenv = (Variable * Tipo) list

type Expr =
    | Num of int
    | Bool of bool
    | Bop of Expr * Operator * Expr
    | If of Expr * Expr * Expr
    | Var of Variable
    | App of Expr * Expr
    | Lam of Variable * Tipo * Expr
    | Let of Variable * Tipo * Expr * Expr
    | Lrec of Variable * (Tipo * Tipo) * (Variable * Tipo * Expr) * Expr

type Value =
    | Vnum of int
    | Vbool of bool
    | Vclos of Variable * Expr * Env
    | Vrclos of Variable * Variable * Expr * Env
    | VRaise
and
    Env = (Variable * Value) list
```

6 Testes

```
1
2  let num01 = Num(1);                          (* OK - TInt *)
3  let num02 = Num(2);                          (* OK - TInt *)
4  let bool01 = Bool(true);                     (* OK - TBool *)
5  let bool02 = Bool(false);                    (* OK - TBool *)
6  let var01 = Var("x");                        (* Fail *)
7  let var02 = Var("y");                        (* Fail *)
8  let op01 = Bop(num01, Sum,num02);            (* OK - TInt *)
9  let op02 = Bop(num01, Gre,num01);            (* OK - TBool *)
10 let op03 = Bop(num01, Sum,var01);            (* Fail *)
11 let op04 = Bop(var01, Sum,var01);            (* Fail *)
12 let if01 = If(bool01,num01,op01);            (* OK - TInt *)
13 let if02 = If(bool02,bool01,op02);           (* OK - TBool *)
14 let if03 = If(num01,num01,num02);            (* Fail *)
15 let if04 = If(bool01,op01,op02);             (* Fail *)
16 let fun01 = Lam("x",TyInt,op01);             (* OK - TFun(TInt,TInt) *)
17 let fun02 = Lam("x",TyInt,op02);             (* OK - TFun(TInt,TBool) *)
18 let fun03 = Lam("x",TyInt,op03);             (* OK - TFun(TInt,TInt) *)
19 let fun04 = Lam("y",TyInt,op03);             (* Fail *)
20 let fun05 = Lam("x",TyInt,op04);             (* Fail *)
21 let app01 = App(fun05,num01);                (* OK - TInt *)
22 let app02 = App(fun05,app01);                (* OK - TInt *)
23 let let01 = Let("x",TyInt,num01,op04);       (* OK - TInt *)
24 let let02 = Let("x",TyInt,num01,op01);       (* OK - TInt *)
25 let let03 = Let("y",TyInt,num01,op04);       (* Fail *)
26
27 let ex1 = Lam ("x", TyInt,
28
29               If( Bop (Var "x", Gre, Num 4),
30                   Bop (Var "x", Sum, Num 1),
31               Num 0));                       (* OK - <x, if(x >= 4) then x + 1 else 0, env>)
32
33 let ex2 = Lrec("fat", (TyInt, TyInt) ,("x", TyInt,
34
35               If(Bop(Var("x"), Eq, Num(0)),
36                   Num(1),
37                   Bop(Var("x"), Mult, App(Var("fat"), Bop(Var("x"), Diff, Num(1)))))
38               ),
39
40               App(Var("fat"), Num(5)))       (* OK - TInt -> 120 *)
```

Todos os testes acima passaram sem problemas pelo nosso avaliador (eval) e pela verificação de tipo (type-check) de acordo com o tipo de resultado esperado!