ソフトウェアサイエンス実験 S8 課題 2-2

200911434 青木大祐

平成 24 年 9 月 14 日

2.2.1 型を見る

以下の式について、REPL で実行して型を確かめた。

```
IntLit 1;;
Plus(IntLit 1, IntLit 3);;
```

リスト 1: 実行結果

```
#     - : exp = IntLit 1
# - : exp = Plus (IntLit 1, IntLit 3)
```

exp 型の式になっていることが分かる。

2.2.2 引き算と割り算の追加

以下のように、exp 型に引き算と割り算を追加した。

```
type exp =
  | IntLit of int          (* リテラル *)
  | Plus of exp * exp      (* 足し算 *)
  | Times of exp * exp     (* かけ算 *)
  | Sub of exp * exp       (* 引き算 *)
  | Div of exp * exp       (* 割り算 *);;
```

2.2.3 式+2

式 E を受け取り、Plus(E + IntLit2) を返す関数を以下のように実装し、式 sample に対して適用した。

```
let func (e:exp) =
  Plus(e, IntLit 2);;

let sample:exp = Div(Plus(IntLit 1, Times(IntLit (-2), IntLit 5)), Sub(IntLit 4, IntLit (-3)));;

func sample;;
```

リスト 2: 実行結果

```
Plus
(Div (Plus (IntLit 1, Times (IntLit (-2), IntLit 5)),
  Sub (IntLit 4, IntLit (-3))),
IntLit 2)
```

2.2.4 絶対値

式の中に出現する整数リテラルを絶対値にする関数を作成した。 再帰的に式を辿って行き、整数リテラルを abs 関数で絶対値にする。

```
let rec abs_exp (e:exp) :exp =
  match e with
    | IntLit i -> IntLit (abs i)
    | Plus(a, b) -> Plus((abs_exp a), (abs_exp b))
    | Times(a, b) -> Times((abs_exp a), (abs_exp b))
    | Sub(a, b) -> Sub((abs_exp a), (abs_exp b))
    | Div(a, b) -> Div((abs_exp a), (abs_exp b));;

abs_exp sample;;
```

リスト 3: 実行結果

```
   Div (Plus (IntLit 1, Times (IntLit 2, IntLit 5)), Sub (IntLit 4, IntLit 3))
```