.1 関数型プログラム言語の体系 (CoreML) における計算の定式化 (EvalDef.v, Eval.v)

この章では、関数型プログラム言語の体系 (CoreML) における計算 (Evaluation) の定式化について (特に講義資料上の表現とシステムでの表現の対応について) 述べる。

項については、前節と同様であるが、念のため、再掲しておく。

講義資料での表現	演習システムでの表現	説明
10	`(% 10)`	整数定数
-13	`(% -13)`	整数定数
`true`	`_true`	真理値定数
`false`	`_false`	真理値定数
$M = N$	`M = N`	比較 (等しさ)
$M > N$	`M > N`	比較 (大なり)
(M, N)	`(M , N)`	対
`left`(M)	`_left M`	左への射影
`right`(M)	`_right M`	右への射影
$\lambda x.M$	`_lambda (x) M`	関数 (ラムダ式)
$M@N$	`M @ N`	関数の適用 (使用)
`if` L `then` M `else` N	`_if L then M else N`	条件式
`let` $x = M$ `in` N	`_let x = M in N`	let 式
`fix` $f.x.N$	`_fix f (x) M`	再帰関数

表 1　項の対応

ほとんどのキーワードの先頭に underscore (_ という記号のこと) がついていることに注意してほしい。ラムダ式と再帰関数において、引数をあらわす `(x)` の括弧は省略できない。

計算規則の表現においては、項の形になっていない「値」や、実行時環境も必要なので、それの表現も与える。

講義資料での表現	演習システムでの表現	説明
10	`(#10)`	値としての整数定数
$true$	`(#true)`	値としての整数定数
`clos`(x, M, E)	`clos(x,M,E)`	関数クロージャ
`rclos`(f, x, M, E)	`rclos(f,x,M,E)`	再帰関数のクロージャ
$[\,]$	`initE` あるいは `[]`	空の実行時環境
$E[x \mapsto v]$	`extend(E,x,v)`	実行時環境

「項としての 10」は `%10` で、「値 (計算結果) としての 10」は `#10` であることに注意されたい。

計算規則の判断 $E \triangleright M \downarrow V$ は、演習システムでは、`E >> M --> V` の形で表現する。たとえば、`initE >> (%10 + %20) --> #30` が成立する。

計算の規則について、講義資料上の表現とシステムでの表現の対応は以下の通りである。

講義資料での表現	演習システムでの表現	説明
var	`Evar`	変数
int	`Eint`	整数定数
bool	`Ebool`	真理値定数
plus	`Eplus (v1) (v2)`	加算
comp1	`Ecomp1 (v1) (v2)`	比較 (大なり)、真になるとき
comp2	`Ecomp2 (v1) (v2)`	比較 (大なり)、偽になるとき
eq1	`Eeq1 (v1) (v2)`	比較 (等しさ)、真になるとき
eq2	`Eeq2 (v1) (v2)`	比較 (等しさ)、偽になるとき
if1	`Eif1`	条件式、真になるとき
if2	`Eif2`	条件式、偽になるとき
pair	`Epair`	対
left	`Eleft`	対の左への射影
right	`Eright`	対の右への射影
lambda	`Elambda`	ラムダ式
fix	`Efix`	再帰関数
apply1	`Eapply1 (x) (t) (E) (v)`	関数適用、関数部分がラムダ式のとき
apply2	`Eapply2 (f) (x) (t) (E) (v)`	関数適用、関数部分が再帰関数のとき
let	`Elet (v)`	let 式

表 2　型付け規則の対応

規則の名前にはすべて大文字の E がついていて、Evar のようなっていることに注意せよ。

Eplus, Ecomp1, Ecomp2, Eeq1, Eeq2 の規則の引数 v1,v2 は、それぞれ引数を計算した結果の値である。(引数ではなく、計算結果であることに注意せよ。つまり、これらの規則を適用するためには、「引数の計算の最終結果が何になるか」をあらかじめ考えなければいけない。)

Eapply1 では、計算したい項が $M@N$ のとき、M を計算した結果は、関数クロージャ $clos(x,t,E)$ の形になることが期待され、また、引数 N を計算した結果は、値 v になることが期待されるが、これらの x,t,E,v を、Eapply1 の引数としてユーザが与える必要がある。

Eapply2 も同様で、計算したい項が $M@N$ のとき、M を計算した結果は、再帰関数クロージャ $rclos(f,x,t,E)$ の形になることが期待され、引数 N を計算した結果は、値 v になることが期待されるが、これらの f,x,t,E,v を、Eapply2 の引数としてユーザが与える必要がある。

Elet では `let` $x = M$ `in` L の形の式を評価するとき，M を計算した結果 V を引数としてユーザが与える必要がある.

Elambda, Eapply1, Eapply2 は長いので，それぞれ Elam, Eapp1, Eapp2 という別名をもっている.

さらにもう 1 つ注意事項であるが、Eplus などの計算で、講義テキストでは、「$(V_1 + V_2 = V$ のとき)」などという条件を書いていた。演習システムでは、これらの条件を明示的に書くことはないが、これらの条件のチェックが残っているときに、演習システムでは、

```
add1 (#10) (#20) = Some (#30)
```

などのゴールが残ってしまうことがある。このような等式がのこっている場合は、(規則に対応するものはないが)、「auto.」 と入力して解消してほしい。なお、

```
add1 (#10) (#20) = Some (#40)
```

のように、成立しない等式が残っている場合は、auto. とやっても、条件が解消されない。このようなときは、もっと前の段階で、規則の適用を誤っていたことになる。

演習システムは、parser については十分こなれていないため、ラムダ式などを入力したとき、予想外の括弧付けをしてしまうことがある。今回の演習では、括弧を多目につけて対処してほしい。