ソフトウェア論理
Logic in Computer Software

亀山幸義

筑波大学 コンピュータサイエンス専攻

第 2 週

前回のまとめ

- ラムダ計算に基づく計算体系 PCF の導入
  - 構文等は、別紙資料参照
  - 今週以降、この計算体系を使って「型システム」について考える。
- 再帰的に定義された関数の意味 (不動点意味論と「たらい」関数の例)

再帰的関数の不動点意味論 (まとめ 1)

再帰的関数:$\texttt{fix}\ f(x).\ \texttt{if}\ x = 0\ \texttt{then}\ 1\ \texttt{else}\ x * (f\ (x-1))$

普通のプログラム言語なら、以下のもの

```
let rec f x =
   if x=0 then 1 else x * f(x-1)
```

直感的意味: 上記関数は以下の (無限) 展開の極限

```
   f x
= if x=0 then 1 else x * f(x-1)
= if x=0 then 1 else x *
   (if x-1=0 then 1 else (x-1) * f(x-2))
= if x=0 then 1 else x *
   (if x-1=0 then 1 else (x-1) *
     (if x-2=0 then 1 else (x-2) * f(x-3))
=...
```


再帰的関数の不動点意味論 (まとめ 2)

- Newton 法
  - $f(x) = 0$ となる実数解 x を探す。
  - 適当な x_0 から始める。
  - n 回目の近似解を x_n とする。
  - $x_{n+1} = x_n - f(x_n)/f'(x_n)$
  - $\lim_{n\to\infty} x_n$ が収束すれば解となる。
- Newton 法を言いかえると
  - $x_0, F(x_0), F(F(x_0)), F(F(F(x_0)))\ldots$ という無限列
  - ただし、$F(x) = x - f(x)/f'(x)$
  - 極限が存在すれば、それが $f(x) = 0$ の解 (の 1 つ)
- 前ページの「直感的意味」は、この「言いかえると」の方に相当する。

再帰的関数の不動点意味論 (まとめ 3)

- 不動点意味論: 再帰的定義による関数 (無限展開の極限) は、以下の「関数を未知数とする方程式」の解となる。

```
f x = if x=0 then 1 else x * f(x-1)
```

- 再帰定理: 上記方程式の解が必ず存在する。
- 不動点意味論: そのうち「最小の解」をとると、プログラム言語における再帰的関数の直感的意味と一致する。
  - Dana Scott: 表示的意味論 (denotational semantics) "Data types as lattices", 1975.

先週の演習問題

```
f(x,y,z) =
 if x<=y then y
 else f(f(x-1,y,z), f(y-1,z,x), f(z-1,x,y))
```

以下では $f(x,y,z)$ を (x,y,z) と省略.

```
(4,2,0)=((3,2,0),(1,0,4),(-1,4,2))=((3,2,0),(1,0,4),4)
=(((2,2,0),(1,0,3),(-1,3,2)),((0,0,4),(-1,4,1),(3,1,0)),4)
=((2,(1,0,3),3),(0,4,(3,1,0)),4)
(3,1,0)=((2,1,0),(0,0,3),(-1,3,1))=((2,1,0),0,3)
(2,1,0)=((1,1,0),(0,0,2),(-1,2,1))=(1,0,2)
(1,0,2)=((0,0,2),(-1,2,1),(1,1,0))=(0,2,1)=2
(2,0,3)=((1,0,3),(-1,3,2),(2,2,0))=((1,0,3),3,2)
(1,0,3)=((0,0,3),(-1,3,1),(2,1,0))=(0,3,2)=3
(2,0,3)=(3,3,2)=3
(3,1,0)=(2,0,3)=3
(4,2,0) =((2,3,3),4,4)=(3,4,4)=4
```


「たらい」関数 (竹内郁雄 1976)

- ヒント:
  f(x,y,z) の計算がどの引数の組合せに対しても停止する (無限ループしない) ことを仮定すれば、f(0,4,f(3,1,0))=4 としてよい (f(3,1,0) を計算する必要はないので、計算量が大幅に減る。)
- f の意味

```
 f(x,y,z)= if x<=y then y else if y<=z then z
else x
```


今週の話題: 型システム

観測: 先週の PCF の構文は「粗っぽい」 (正当なプログラムとして認めたくないものまで、構文上はプログラムとして許している)

- 関数の位置に数が来ることがある。

$$(10\ \texttt{true}) + 30$$

- 真偽値と数を比較することがある。

$$(\lambda x.\ \texttt{if}\ x = 10\ \texttt{then}\ 20\ \texttt{else}\ 30)\ \texttt{false}$$

問題: どういうプログラム e が、このようなエラーを引きおこすかを計算実行前に知りたい。

型システム (Type System)

- 「良い」プログラムかどうかを判別する手段
- プログラム e に型がつけば $e \not\rightsquigarrow \texttt{wrong}$ とは決してならない。(ただし、wrong はこの種のエラーをあらわす特別な値)
- 型がつくかどうかは、コンパイル時 (実行前) に判定。
- → 誤りを早期に発見できる。有効なディバッグ方法。
- おまけ: コンパイラは、型情報を使って、より高速に動作するコードを生成できる可能性がある。
- おまけ: プログラムを読む人にとって、型情報は良いドキュメントとなる。(保守、再利用、検証 etc.)

実行前 (コンパイル時) = 静的 static
実行時 = 動的 dynamic

型付け

型付けの例:

$10 + 20 : \texttt{int}$

$\texttt{if true then } 10 \texttt{ else } 20 : \texttt{int}$

$\lambda x.\ \texttt{if true then } x + 10 \texttt{ else } 20 : \texttt{int} \to \texttt{int}$

$\lambda (x, y).\ \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : (\texttt{int}, \texttt{bool}) \to \texttt{int}$

$\lambda y.\ \lambda x.\ \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{bool} \to (\texttt{int} \to \texttt{int})$

型付け

例:

$\texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$\texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$(\texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20) + (\texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20) : \texttt{int}?$

項が変数 (正確には「自由変数」) を持つとき、その変数の型をどこかで覚えておかないとうまく行かない。

型付け

$x : \texttt{int}, y : \texttt{bool} \vdash \texttt{if } y \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

$x : \texttt{int}, y : \texttt{int} \vdash \texttt{if } y = 30 \texttt{ then } x + 10 \texttt{ else } 20 : \texttt{int}$

一般に、型判定 (Judgment) は以下の形をとる。

$$\Gamma \vdash e : \tau$$

ここで、Γ は $x_1 : \sigma_1, \cdots, x_n : \sigma_n$ の形をしている。
e は項、τ は型。

型システムの規則-1

$$\frac{((x:\sigma)\in\Gamma\text{のとき})}{\Gamma\vdash x:\sigma}\ \text{var}$$

$$\frac{(m\text{ が整数定数のとき})}{\Gamma\vdash m:\texttt{int}}\ \text{const1}$$

$$\frac{(b\text{ が }\texttt{true}\text{ か }\texttt{false}\text{ のとき})}{\Gamma\vdash b:\texttt{bool}}\ \text{const2}$$

型システムの規則-2

$$\frac{\Gamma\vdash e_1:\texttt{int}\quad\Gamma\vdash e_2:\texttt{int}}{\Gamma\vdash e_1+e_2:\texttt{int}}\ \text{plus}$$

$$\frac{\Gamma\vdash e_1:\texttt{bool}\quad\Gamma\vdash e_2:\sigma\quad\Gamma\vdash e_3:\sigma}{\Gamma\vdash\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3:\sigma}\ \text{if}$$

型システムの規則-3

$$\frac{\Gamma,x:\sigma\vdash e:\tau}{\Gamma\vdash\lambda x.\,e:\sigma\to\tau}\ \text{fun}$$

$$\frac{\Gamma\vdash e:\sigma\to\tau\quad\Gamma\vdash f:\sigma}{\Gamma\vdash e\ f:\tau}\ \text{apply}$$

型システムの規則-4

$$\frac{\Gamma,f:\sigma\to\tau,x:\sigma\vdash e:\tau}{\Gamma\vdash(\texttt{fix}\ f(x).\,e):\sigma\to\tau}\ \text{fix}$$

$\texttt{fix}(f)(x).\,e$ が $f(x)=e$ となるような関数 f のこととわかってしまえば、型付け規則も簡単に理解できる.

型の導出の例

$$\dfrac{\dfrac{}{\cdot \vdash 0 : \mathtt{int}} \quad \dfrac{}{\cdot \vdash 1 : \mathtt{int}}}{\cdot \vdash 0 + 1 : \mathtt{int}}$$

型の導出の例

Let $\Gamma = x : \mathtt{int}, y : \mathtt{bool}$.

$$\dfrac{\dfrac{\dfrac{}{\Gamma \vdash y : \mathtt{bool}} \quad \dfrac{}{\Gamma \vdash x : \mathtt{int}} \quad \dfrac{\dfrac{}{\Gamma \vdash x : \mathtt{int}} \quad \dfrac{}{\Gamma \vdash 1 : \mathtt{int}}}{\Gamma \vdash x + 1 : \mathtt{int}}}{\Gamma \vdash \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{int}}}{y : \mathtt{bool} \vdash \lambda x.\ \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{int} \to \mathtt{int}}}{\cdot \vdash \lambda y.\ \lambda x.\ \mathtt{if}\ y\ \mathtt{then}\ x\ \mathtt{else}\ x + 1 : \mathtt{bool} \to (\mathtt{int} \to \mathtt{int})}$$

型の導出の例

Let $\Delta = x : \mathtt{bool} \to \mathtt{bool}, y : \mathtt{bool}$.

$$\dfrac{\dfrac{\dfrac{\dfrac{}{\Delta \vdash x : \mathtt{bool} \to \mathtt{bool}} \quad \dfrac{\dfrac{}{\Delta \vdash x : \mathtt{bool} \to \mathtt{bool}} \quad \dfrac{}{\Delta \vdash y : \mathtt{bool}}}{\Delta \vdash xy : \mathtt{bool}}}{\Delta \vdash x(xy) : \mathtt{bool}}}{x : \mathtt{bool} \to \mathtt{bool} \vdash \lambda y.\ x(xy) : \mathtt{bool} \to \mathtt{bool}}}{\cdot \vdash (\lambda x.\ \lambda y.\ x(xy)) : (\mathtt{bool} \to \mathtt{bool}) \to (\mathtt{bool} \to \mathtt{bool})}$$

型システムの規則-5

$$\dfrac{\Gamma, x_1 : \sigma_1, \cdots, x_n : \sigma_n \vdash e : \tau}{\Gamma \vdash \lambda(x_1, \ldots, x_n).\ e : (\sigma_1, \ldots, \sigma_n) \to \tau}\ \mathrm{fun2}$$

$$\dfrac{\Gamma \vdash e : (\sigma_1, \ldots, \sigma_n) \to \tau \quad \Gamma \vdash e_1 : \sigma_1 \quad \ldots \quad \Gamma \vdash e_n : \sigma_n}{\Gamma \vdash e_0\ (e_1, \ldots, e_n) : \tau}\ \mathrm{apply2}$$

型システムの規則-6

$$\frac{\Gamma,\ f : (\sigma_1, \ldots, \sigma_n) \to \tau,\ x_1 : \sigma_1, \ldots, x_n : \sigma_n \vdash e : \tau}{\Gamma \vdash (\texttt{fix}\ f(x_1, \ldots, x_n).\ e) : (\sigma_1, \ldots, \sigma_n) \to \tau}\ \text{fix}$$

$\texttt{fix}\ f(x_1, \ldots, x_n).\ e$ が $f(x_1, \ldots, x_n) = e$ となるような関数 f のこと.

Exercise

- $e + f$ の形の項に対する規則を参考にして、$e - f$ や $e = f$ の形の項に対する型の導出規則を定めなさい。
- 型判断 $x : \texttt{int} \to \texttt{int}, y : \texttt{int} \vdash x(x\ y)) : \texttt{int}$ に対する導出を書きなさい。
- 適当な σ, σ', τ に対して、型判断 $x : \sigma,\ y : \tau \vdash x(x\ y) : \sigma'$ の導出があるか調べなさい。

課題

1. fact 関数の型を導きなさい。
2. 項 $\lambda x.\ x\ x$ は型がつくか？

型システムの健全性 (Type Soundness)

- プログラミング言語の型システムが「きちんとできている」ことを意味する性質。
- 1 つ 1 つのプログラムの性質ではなく、プログラミング言語の性質。
- Subject Reduction (Preservation) と Progress の 2 つの性質を合わせたもの。

Theorem (Subject Reduction)

$\Gamma \vdash e : \sigma$ を導くことができて、$e \rightsquigarrow^* e'$ であるならば、$\Gamma \vdash e' : \sigma$ を導くことができる。

ただし $\rightsquigarrow^*$ は $\rightsquigarrow$ (1 ステップの計算) の反射的推移的閉包で、「0 以上のステップの計算」をあらわす。

型システムの健全性 (Type Soundness)

- プログラミング言語の型システムが「きちんとできている」ことを意味する性質。
- 1 つ 1 つのプログラムの性質ではなく、プログラミング言語の性質。
- Subject Reduction (Preservation) と Progress の 2 つの性質を合わせたもの。

Theorem (Subject Reduction)

$\Gamma \vdash e : \sigma$ を導くことができて、$e \rightsquigarrow^* e'$ であるならば、$\Gamma \vdash e' : \sigma$ を導くことができる。

- プログラムの実行中に、プログラムの型が保存されることを意味する。(Preservation ともいう)
- 型の整合性が崩れない、ことも意味する。
- 自由変数が増えないことも意味する。

本日のまとめ

ふだん何気なく使っているプログラム言語の型システムの役割を考えた。