組込みシステム向けFRP言語に対する第一級関数の導入

横山陽彦・渡部卓雄（東京工業大学）

概要

・小規模組込みシステム向け関数リアクティブプログラミング(FRP)言語に対し，制限された第一級関数を導入した
・高階関数を扱う言語EFλと一階の関数を扱う言語EFcoreを定義し，前者から後者への変換を示した
・実行時におけるオブジェクト生成数の上限を静的に決定しつつ，第一級関数を扱えるという要請を満たした

既存研究：Emfrp [Sawada et al 2016]

関数リアクティブプログラミング(FRP)とは，時変値と呼ばれる時刻とともに変化する値を組み合わせ，GUIなどの反応的な処理の記述を行うプログラミング手法である．
弊研究室で開発されたEmfrpは，以下のような機能を持つ．
・小規模組込みシステム向けFRP言語
・ノード(時変値)を組み合わせて処理を記述
・MLライクな型システムを持ち，代数的データ型が使用可能
・高階関数，再帰的なデータ型定義，再帰関数定義の禁止
・実行時の使用オブジェクト生成数を静的に決定可能
Emfrpでは，記述に関する制約が多く冗長なプログラムになってしまうことがある．

```
module PosX    # mudule
in vl : Float, # left
   vr : Float  # right
   th : Float  # theta
out x : Float  # pos
use Std

node i =
    (vl + vr) * cos(th)
node init[0] x =
    0.5 * (x@last + i)
```

Emfrpでの記述例

$$x(t) = \frac{1}{2}\int_0^t (v_l(t) + v_r(t))\cos(\theta(t))dt$$

時刻に依存した座標

```
type V3[a] = V3(a, a, a)

func clamp(a, b, x) =
  if x < a then a else
  if x > b then b else
  x

func clamp3(a, b,
            v: V3[Int]) =
  v of V3(v1, v2, v3) ->
    V3(clamp(a, b, v1),
       clamp(a, b, v2),
       clamp(a, b, v3))
```

Emfrpでの冗長な記述例(抜粋)

提案言語：EFλ

Emfrpをベースに，制限された第一級関数を導入したFRP言語
・MLライクな型システム
・fun式により第一級関数を定義できる
・データ型やノードに関数型は含められない
・再帰的なデータ型，再帰関数は禁止
・関数型に対して一部制限がある

Algebraic Data Type

$$D_\chi \quad ::= \quad \texttt{type}\ \chi[\delta,\ldots,\delta] = d(\tau^d,\ldots,\tau^d) \mid \ldots \mid d(\tau^d,\ldots,\tau^d)$$

$$\frac{f : \forall\delta_1,\ldots,\delta_n.(\tau_1,\ldots,\tau_m) \to \tau \in \Gamma \quad \theta = [\delta_1 \mapsto \tau_1^d,\ldots,\delta_n \mapsto \tau_n^d]}{\Gamma \vdash f[\tau_1^d,\ldots,\tau_n^d] : \theta((\tau_1,\ldots,\tau_m) \to \tau)} \quad \text{(T-FUNC)}$$

$$\frac{\Gamma, x_1{:}\tau_1,\ldots,x_n{:}\tau_n \vdash e : \tau}{\Gamma \vdash \texttt{fun}\ (x_1{:}\tau_1,\ldots,x_n{:}\tau_n)\ \texttt{->}\ e : (\tau_1,\ldots,\tau_n) \to \tau} \quad \text{(T-ABS)}$$

$$\frac{\Gamma \vdash e : (\tau_1,\ldots,\tau_n) \to \tau \qquad \Gamma \vdash e_1 : \tau_1,\ldots,\Gamma \vdash e_n : \tau_n}{\Gamma \vdash e(e_1,\ldots,e_n) : \tau} \quad \text{(T-APPLY)}$$

$$\frac{\Gamma \vdash e_1 : \tau_1 \qquad \Gamma, x : \tau_1 \vdash e_2 : \tau}{\texttt{let}\ x = e_1\ \texttt{in}\ e_2 : \tau} \quad \text{(T-LET)}$$

$$\frac{\Gamma \vdash e_1 : \texttt{Bool} \qquad \Gamma \vdash e_2 : \tau^d \qquad \Gamma \vdash e_3 : \tau^d}{\texttt{if}\ e_1\ \texttt{then}\ e_2\ \texttt{else}\ e_3 : \tau^d} \quad \text{(T-IF)}$$

```
try Error = E_X | E_Y | E_Z
type R[a] = Ok(a) | Err(Error)

func try[a, b](x: R[a],
               f: (a)->R[b]):R[b] =
  case x of
  | Ok(k) -> f(k)
  | Err(e) -> Err[b](e)

func calc_sum(rx: R[Int],
              ry: R[Int],
              rz: R[Int]): R[Int] =
  try[Int, Int](rx, fun (x:Int) ->
  try[Int, Int](ry, fun (y:Int) ->
  try[Int, Int](rz, fun (z:Int) ->
    Ok[Int](x + y + z)
  )))

input sensor_x: R[Int]
input sensor_y: R[Int]
input sensor_z: R[Int]

node sum: R[Int] =
  calc_sum(sensor_x,
           sensor_y,
           sensor_z)
```

EFλでの高階関数tryを用いた記述例

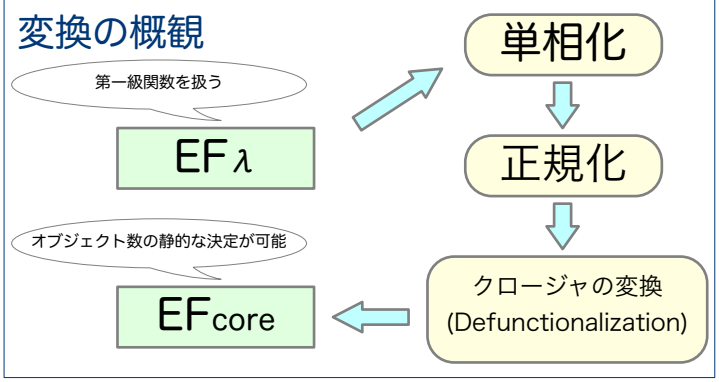
提案言語：EFcore

Emfrpのサブセット言語
・単相の型システム
・一階の関数のみを扱う
・オブジェクト生成数の静的な決定が可能

```
func init_x((): Unit): Float = 0.0
func calc_i(vl: Float, vr: Float,
            th: Float): Float =
  let tmp1 = vl + vr in
  let tmp2 = cos(th) in
  tmp1 * tmp2
func calc_x(xl: Float, i: Float): Float =
  let tmp = xl + i in
  0.5 * tmp

input vl : Float
input vr : Float
input th : Float
node i = calc_i(vl, vr, th)
node x init init_x(()) = calc_x(x@last, i)
```

変換の概観



変換例

```
...

type V3[a] = V3(a, a, a)

func map3[a, b](v: V3[a],
                f:(a) -> b): V3[b] =
  case v of V3(x1, x2, x3) ->
    V3[b](f(x1), f(x2), f(x3))

func clamp(...): Int = ...

func clamp3(a: Int, b: Int,
            v: V3[Int]): V3[Int] =
  map3[Int, Int](v,
    fun (x: Int) -> clamp(a, b, x))

...
```

EFλでの高階関数mapを用いた記述例(抜粋)

変換されたEFcore

```
type V3_Int = V3_Int(Int, Int, Int)
type _E1 = _E1(Int, Int)

func clamp(...): Int = ...

func _E1func(_e: _E1, x: Int): Int =
  case _e of _E1(a, b) ->
    clamp(a, b, x)

func map3_Int_Int_E1(v: V3_Int,
                     f: _E1): V3_I =
  case v of V3_Int(x1, x2, x3) ->
    let _t1 = _E1func(f, x1) in
    let _t2 = _E1func(f, x2) in
    let _t3 = _E1func(f, x3) in
    V3_Int(_t1, _t2, _t3)

func clamp3(a: Int, b: Int,
            v: V3_Int): V3_Int =
  let _t = _E1(a, b) in
  map3_Int_Int_E1(v, _t)
```

実装・実験

・時間的，空間的負荷を測定
・実行環境：ATmega328p (CPU 16MHz / ROM 32KB / RAM 2KB)

言語	.text[byte]	.data+.bss [byte]	実行時間 [ms]
EFλ	4138	262	2025
Emfrp	3956	254	757

map関数を用いた記述の測定結果

言語	.text[byte]	.data+.bss [byte]	実行時間 [ms]
EFλ	4652	302	3416
Emfrp	4412	278	3239

try関数を用いた記述の測定結果