関数リアクティブプログラミング言語による小規模組込みシステムのプログラミング

渡部卓雄

東京工業大学 情報理工学院 情報工学系

SWEST23 (2021/9/2)

関数リアクティブプログラミング(FRP)

- 関数プログラミングの考え方にもとづいて，リアクティブシステムの記述を支援するプログラミングパラダイム
  - リアクティブシステム：外界から連続的・非連続的に与えられる入力に反応し続けるシステムのこと．入力のタイミングや順番は予測できない．
    - 例：GUI，組込みシステム，サイバーフィジカルシステム(CPS)
    - 対義語：変換的(transformational)システム．有限な入力に対して答を出力して停止する（古典的な計算モデルにもとづく）システム．
  - FRP言語の例：Fran [Elliot '97], Yampa [Hudak '03], Elm [Czaplicki, '12], etc.
- ポイント：時間とともに変化する値を時変値(Time-Varying Value)として表現し，その上の計算としてリアクティブな動作を記述する．

Emfrp

本研究室で開発した組込みシステム向けFRP言語[1]

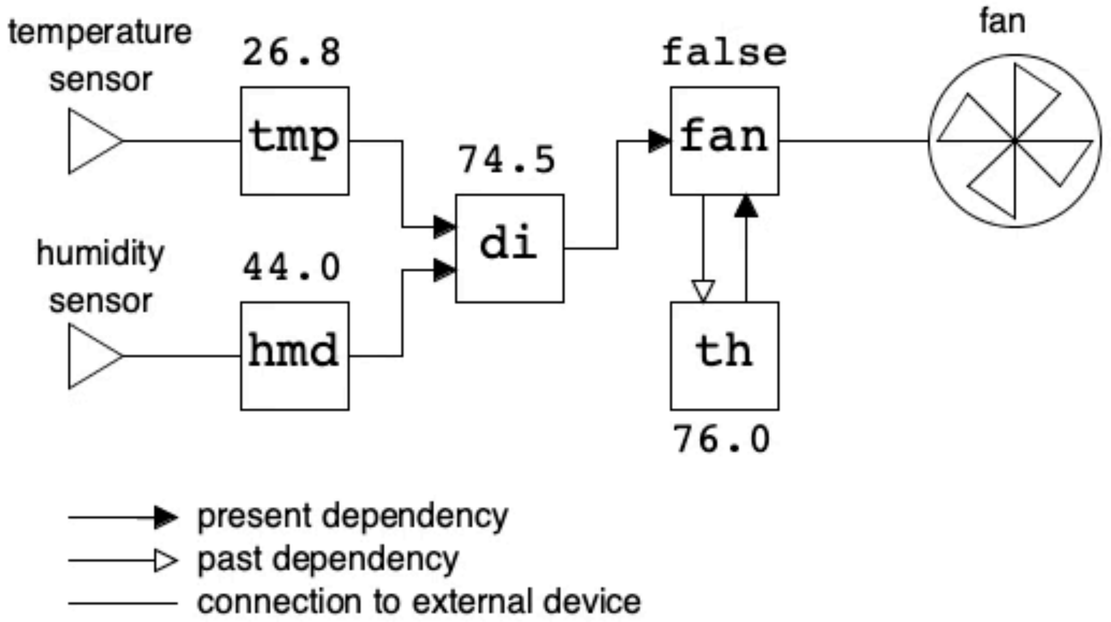
Emfrpによるファン制御プログラムの例

```
module FanController
in  tmp : Float  # 温度センサ
    hmd : Float  # 湿度センサ
out fan : Bool   # ファンのスイッチ
use Std

node di = 0.81 * tmp + 0.01 * hmd
     * (0.99 * tmp - 14.3) + 46.3;

node init[False] fan = di >= th

node th = 75.0 +
  if fan@last then -1.0 else 1.0
```



Emfrpのプログラムは，時変値を節点，それらの依存関係を有向辺とする有向グラフとして図式化できる．fanからthへの辺は，thの定義におけるfanの直前値の参照（プログラム中のfan@last）を表している．この仕組みにより，Emfrpは副作用を持たない関数型言語にもかかわらず状態を持つ動作を記述できる．

プログラム中のtmp, hmd, di, fan, th は時変値であり，時間と共に変化する．
この例ではファンのモータを保護するためのヒステリシス制御も行っている．

処理系: github.com/psg-titech/emfrp
あるいは gem install emfrp

[1] Sawada, K. & T. Watanabe, "Emfrp: A Functional Reactive Programming Language for Small-Scale Embedded Systems", CROW 2016, ACM, 2016.

倒立振子ロボットの記述例

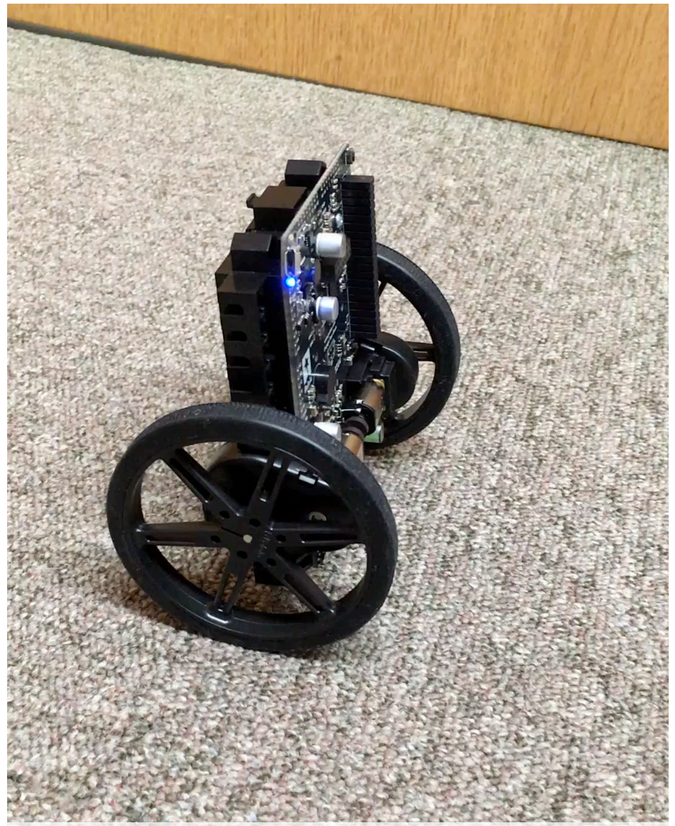
```
module Balancer
in  gyroY  : Int,  # gyroscope (y-axis)
    accX   : Int,  # accelerometer (x-axis)
    encL   : Int,  # left motor encoder
    encR   : Int   # right motor encoder
out motorL : Int,  # left motor
    motorR : Int   # right motor
use Std
...                # definitions of constants

node init[0] angle =
    (angle@last + gyroY * update_time_ms) * 99 / 100

node speedL = encL - encL@last
node speedR = encR - encR@last
node init[0] distL = distL@last + speedL
node init[0] distR = distR@last + speedR

node risingAngleOff  = gyroY * angle_rate_ratio + angle
node init[0] motor =
    motorSpeed(motor@last +
                (angle_resp * risingAngleOff +
                 dist_resp * (distL + distR ) +
                 speed_resp * (speedL + speedR)) / 100 / gear_ratio)

node diffSpeed = (distL - distR) * dist_diff_resp / 100
node motorL = if accX > 0 then motor + diffSpeed else 0
node motorR = if accX > 0 then motor - diffSpeed else 0
```



Pololu Balboa 32U4
(ATmega 32U4, 32KB Flash, 2.5KB RAM)

github.com/psg-titech/emfrp_samples

Emfrpの特徴

- 純粋な関数リアクティブプログラミング言語
  - 静的型，型推論，代数的データ型
  - コールバックやリフティングが不要（時変値と普通の値を混在できて直感的に書ける）
- マイクロコントローラなどの小規模組込みシステム向けに設計
  - ソースコードは標準的なCにコンパイルされる
    - プラットフォーム依存の部分とリンクすることで実行可能コードを生成
  - 実行時のメモリサイズが静的に決定される
    - 実行中にメモリ不足になることはない
  - 時変値の更新に用いる式の計算の停止性を保証
    - 実行中にハングして無反応になることはない

- 時変値を一級データとしない
- データ型・関数定義における再帰を禁止

$\text{Emfrp}^{\text{BCT}}$：再帰的データ構造の導入

- 組込みにおいても再帰的データ構造と再帰関数は必要：木，リスト，etc.
- Emfrpの良い性質は残したい
  - 実行時メモリサイズの静的確定・時変値更新処理の停止性

- 提案手法：BCT (Bounded Construction Types)
  - データの大きさを規定できる型システム
    - 先行研究(Sized Types)より精密な大きさ指定が可能
  - $\text{Emfrp}^{\text{BCT}}$：BCTを導入したEmfrpの拡張

型付け規則

$$\Gamma \vdash_f^{\mathcal{T};\mathcal{F}} e : \tau | C$$

操作的意味論

$$[s]E | [t]H \vdash_{\mathcal{L}}^{\mathcal{T};\mathcal{F}} e \Downarrow_u l; [t']H'$$

[2] Yokoyama, A., S. Moriguchi & T. Watanabe, "A Functional Reactive Programming Language for Small-Scale Embedded Systems with Recursive Data Types", J. of Information Processing, IPSJ, Nov., 2021 (掲載予定).

XStorm：状態依存動作の記述支援

- 組込みシステムの動作は状態遷移系として記述されることが多い
  - 例）階層的状態遷移系にもとづくモデルベース開発
- 副作用のない純粋FRPにおける状態の表現
  - EmfrpではX@lastで時変値Xの直前の値を参照できるので，この機構を用いて状態を表現
    - 状態依存の動作を書くために各時変値の定義に条件式が必要　⇒コードが煩雑になる
  - 文脈指向プログラミング(COP)の導入[3]：
    - 状態依存動作は簡潔に書けるが，状態遷移系の表現は面倒
- XStorm: 状態遷移系を陽に記述する機構を導入したFRP言語[4]
  - アプリケーションのモデル検査の支援も可能[5]

[3] Watanabe, A Simple Context-Oriented Programming Extension to an FRP Language for Small-Scale Embedded Systems, COP '18, ACM, 2018.
[4] 松村ほか, 組込みシステム向けFRP言語における状態依存動作のための抽象化機構, 情報処理学会論文誌（プログラミング）, 13(2), pp. 1-13, 2020.
[5] 内藤ほか, 組込みシステム向けFRP言語におけるモデル検査を用いた状態依存動作の検証, SIGEMB57, 2021.

他のプロジェクト

- 並列・分散環境への対応
  - **マルチコア実行**：Sakurai, et al, Towards a Statically Scheduled Parallel Execution of an FRP Langauge for Embedded Systems, REBLS '21, ACM, 2021.
  - **GPGPUによる実行**：Sakurai, et al, Functional Reactive Programming for Embedded Systems with GPGPUs, ICSCA '21, ACM, 2021.
  - **分散実行**：Shibanai, et al. Distributed Functional Reactive Programming on Actor-Based Runtime, AGERE '18, ACM 2018.
- 実時間処理
  - **周期的タスクの記述**：辻ほか, 小規模組込みシステム向けFRP言語における周期的タスクの記述方式, ソフトウェア科学会大会, 2021.
  - **時間制約付きイベント記述**： 堀ほか, 関数リアクティブプログラミング言語のための時間制約付きイベントの記述方式, SIGEMB57, 2021.
- 安全性・形式手法
  - **安全な初期化**：白鳥ほか 関数リアクティブプログラミングにおける時変値の初期化手法の提案, ソフトウェア科学会大会, 2021.
- 応用
  - **WSANへの応用**：後藤ほか, WSAN向けマクロプログラミング言語の提案, コンピュータソフトウェア, 38(2), 2021.

なぜ新しい言語なのか？

- 構文や型による（強力な）制約が欲しい
  - 「この言語で書いたプログラムではこういう性質が常に成立する」と言いたい
    - メモリ制約，停止性，実時間性，etc.

- 「C++ / Rust / Python / Haskell, etc の拡張ではダメなの？」
  - 時変値（あるいはシグナル関数）を既存言語のオブジェクトや関数として表現する必要
    - x = y + z を x = app2 lift2(+) (y, z) のように書きたくない
  - 新しいアイディアが既存の言語上で必ずしも素直に表現できるとは限らない

将来課題

- 形式手法の導入
  - 安全性・実時間性の静的検証
- 連続値のより適切な扱い
  - 制御・信号処理の知見を活用?
- 開発・デバッグ支援
  - デバッガ，ビジュアルコーディング，ライブコーディング環境
- FPGA用の高レベル記述言語としてのFRP
- 研究成果を取り入れた改良版処理系
  - 状態依存動作，状態遷移記述のサポート，再帰の導入，マルチコア対応など