Vol.2019-EMB-51 No.2
2019/6/21

組込みシステム向けFRP言語における動的動作のための抽象化機構

松村 有倫[1,a)]　渡部 卓雄[1,b)]

概要：メモリ資源に制約のある小規模組込みシステムを対象とした関数リアクティブプログラミング (FRP) 言語である Emfrp に対し，システムの状態に応じた計算の動的な切り替えを行うための抽象化機構の導入を提案する．状態ごとに時変値と次状態の更新計算を記述できる機構を提供することで，FRP の枠組みの中で状態遷移を扱うことを可能にしている．この抽象化機構の導入によって，振る舞いが動的に変化するコンポーネントの表現が容易になることを例を通して示す．

1. はじめに

外部からの入力への応答と自身の状態の更新を続けるリアクティブシステムの典型例として組込みシステムを挙げることができる．関数リアクティブプログラミング (Functional Reactive Programming, FRP) は，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である．時間変化する値である**時変値**を組み合わせて処理を記述することで，入力に対する応答を宣言的に記述することができる．

マイクロコントローラなどの計算資源の乏しい小規模組込みシステムにおけるリアクティブシステムの開発を支援するため，我々は FRP 言語 Emfrp[1] や XFRP[2] の設計・実装を行ってきた．言語の構文にいくつかの制約を設けることによってメモリ消費量を静的に決定できる設計になっている．具体的には，関数やデータ型の再帰的な定義の禁止や，無名関数を利用できる構文の制限によって動的にメモリを確保することなく動作することができる．また，push 型と呼ばれる実行方式を採用し，時変値の過去の値の参照を直前の値に制限することで空間漏れ・時間漏れの問題を解決している．これらの言語は小規模組込みシステムにおけるリアクティブシステムの宣言的な記述手法を提供しているが，上述の制約によって実行時に振る舞いが変化するような処理の記述を苦手としている．状態遷移によって設計されたシステムの実装もこれに該当する．

FRP において動的に振る舞いを変化させるための抽象化機構としては switch オペレータが知られている．これは Haskell の FRP ライブラリである Yampa[3] などに導入されている時変値上の関数の時変値を扱うオペレータである．時変値に応じて適用する関数を切り替えることによって時変値の更新計算を実行時に変化させることができるが，動的なメモリの確保を行わずに動作するという制約のもとでこのようなオペレータを導入するのは困難である．また，適応動作記述のモジュール性向上を目的として，我々は Emfrp への文脈指向プログラミング機構の導入 [4] を提案している．これによって実行時情報に依存する動作をモジュール化して記述できるようになり，変化する環境に適応してシステムの状態を変化させることが可能となった．しかし，動作の切り替えは実行時情報に対する条件式の形で記述するため，複雑な状態遷移の表現を行うことは難しい．

システムは状態遷移図を用いて設計されることが多い．UML のステートマシン図は状態における動作と状態の遷移によって記述され，ある状態からの遷移は遷移条件と遷移先状態によって表現される．各状態ごとにその状態における動作と遷移先状態を求める計算を記述できる機構があれば状態遷移を含むシステムを表現することが可能になる．

こうした抽象化機構として，本研究では XFRP に対する Switch 拡張を提案する．この拡張は先に挙げた switch オペレータのアイデアを XFRP に導入するもので，実行時における時変値の更新計算の切り替えを可能にする．Switch 拡張では各状態に対して時変値と次状態の更新計算を記述し，実行時に現在の状態に応じてそれらを切り替える．一般の switch オペレータと比較すると，切り替える先の計算が静的に決定するため，動的なメモリの確保を行わずに動

[1] 東京工業大学 情報理工学院 情報工学系
Department of Computer Science, School of Computing, Tokyo Institute of Technology
[a)] arimichi@psg.c.titech.ac.jp
[b)] takuo@acm.org

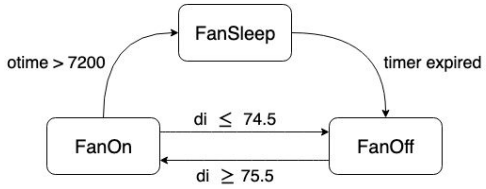

図 1 扇風機制御プログラムの状態遷移

作するという性質を保ちつつ実現することができる．

2. XFRP に対する Switch 拡張

この章では，扇風機の制御プログラムの例を通じて XFRP の Switch 拡張について説明する．この例では，時間変化する不快指数をもとに扇風機のスイッチを制御している．この他に，状態に依存した動作として (a) スイッチの現状態に応じたヒステリシス制御機能，(b) 累積稼働時間および連続稼働時間を計算する機能，(c) 連続稼働時間が 2 時間を超えた際に 30 分のスリープを入れる機能が追加されている．

今回の例では，このシステムをオフ状態，オン状態，スリープ状態の 3 状態に分けて記述している．**図 1** に状態遷移図を示す．(a) については状態に応じて遷移の閾値を変えることで，(b) については状態ごとに時変値の更新計算を切り替えることで，(c) についてはスリープ状態を導入することで実現している．

図 2 は，制御プログラムの概略である．冒頭部にて入出力時変値や初期状態などの宣言を行い（1-8 行目），続けて `state` ブロックの中に各状態の定義を記述する．**図 3** は，オン状態の記述例である．1 行目に記述されている `begin` は状態のパラメータであり，ここではオン状態へ遷移した時刻を表している．ブロック内部では宣言した出力時変値の更新計算を記述し，11 行目から始まる `switch` 節に次状態の計算を記述している．`Retain` は現在の状態を表すキーワードである．

Swtich 拡張を用いずに同様の記述を行う場合，状態に応じた処理の変化は条件分岐によって実現することになる．状態に対する条件分岐が様々な時変値の定義に散らばってしまうことで可読性や拡張性の低下を招いてしまう．提案する手法では状態に応じた振る舞いの変化が各状態ごとにモジュール化されるため，状態に依存する動作の把握が容易になる．また，状態の追加を行う際の既存の実装への影響も小さくなる．

```
1  switchmodule FanState # module name
2  in    clock(0) : Int,  # elapsed seconds
3        di       : Float # discomfort index
4  out   ctime(0) : Int,  # cumulative op. time
5        otime    : Int,  # continuous op. time
6        fan      : Bool  # fan switch
7  use Std                # standard library
8  init FanOff            # initial state
9
10 # when the fan is OFF
11 state FanOff {
12   ...
13 }
14
15 # when the fan is ON
16 state FanOn(begin : Int) {
17   ...
18 }
19
20 # when the fan sleep
21 state FanSleep(begin : Int) {
22   ...
23 }
```

図 2 Switch 拡張を用いた扇風機制御プログラム

```
1  state FanOn(begin : Int) {
2    node ctime =
3      let dt = clock - clock@last in
4      ctime@last + dt
5
6    node otime = clock - begin
7
8    node fan = True
9
10   # next state transition
11   switch :
12     if di <=. 74.5 then
13       FanOff
14     else if otime > 7200 then
15       FanSleep(clock)
16     else Retain
17 }
```

図 3 扇風機が ON の状態に対する記述例

参考文献

[1] Sawada, K. and Watanabe, T.: Emfrp: A Functional Reactive Programming Language for Small-scale Embedded Systems, *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, New York, NY, USA, ACM, pp. 36–44 (online), DOI: 10.1145/2892664.2892670 (2016).

[2] Watanabe, T. and Sawada, K.: Towards Reflection in an FRP Language for Small-Scale Embedded Systems, pp. 1–6 (online), DOI: 10.1145/3079368.3079387 (2017).

[3] Hudak, P., Courtney, A., Nilsson, H. and Peterson, J.: Arrows, Robots, and Functional Reactive Programming, *Advanced Functional Programming* (2002).

[4] Watanabe, T.: A Simple Context-Oriented Programming Extension to an FRP Language for Small-Scale Embedded Systems, *Proceedings of the 10th International Workshop on Context-Oriented Programming: Advanced Modularity for Run-time Composition*, COP '18, New York, NY, USA, ACM, pp. 23–30 (online), DOI: 10.1145/3242921.3242925 (2018).