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　渡部 卓雄（東京工業大学）

計算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．

関数リアクティブプログラミング(FRP)

時間変化する入力に応答して計算処理を行うシステムをリアクティブシステムと呼ぶ．典型的には，ユーザーからの入力に応答して処理を行うUIや，センサーの値に応答して制御を行うコントローラなどが挙げられる．関数リアクティブプログラミング(FRP)は，時間変化する値である時変値を扱うことでリアクティブシステムの宣言的な記述を支援するプログラミングパラダイムである．入力に対する応答は依存する時変値の変化に伴う値の再計算という形で実現される．

Emfrpに対するSwitch拡張

多くの組込みシステムは複雑な状態遷移を含み，それらの開発においては状態に応じて振る舞いを変化させるためにプログラムが複雑になる．本研究では小規模組込みシステムを対象としたFRP言語であるEmfrp[Sawada et al. 2016]に対してswitchオペレータに相当する拡張（Switch拡張）を行うことで，状態遷移を含むシステムの宣言的な記述を可能にする．switchはHaskellのFRPライブラリであるYampa[Hudak et al. 2003]におけるオペレータで，時変値計算の切り替えを動的に行うことができる．Yampaのswitchオペレータとは異なり，Switch拡張では切り替え先の状態の集合が静的に決定される．そのため，動的なメモリ確保を行うことなく動作させることができ，マイクロコントローラなどの計算資源の乏しい環境にも適用することができる．

```
switchmodule Example # module name
in   input1 : Int,     # input
     input2 : Int      # input
out output : Bool      # output
use Std                # library
init StateA            # initial state

# state definition
state StateA {
  node output =
    (input1 + input2) % 2 == 0
  # next state
  switch:
    if pred then
      StateB(0)
    else Retain
}
…
```

入力時変値と出力時変値の形式，初期状態などを記述する．

状態を定義し，その状態における時変値の計算を記述する．状態に対してパラメータを設けることもできる．

現在の状態から他の状態への遷移を記述する．

ケーススタディ

例題として，カウントアップ機能とカウントダウン機能のついたタイマーの実装を考える．このタイマーは3つのボタンと時，分，秒の桁があるディスプレイを持ち，その動作は図1のような状態遷移図で表すことができる．図2,3はSwitch拡張を用いた実装例の概略である．状態遷移図中の各状態に対して時変値計算の定義と次状態への遷移を記述している．カウントダウンタイマーを設定する状態であるCountDownSetについては操作している桁をパラメータとして持たせることで処理の共通する状態をまとめている．

Switch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モジュール化されるため，状態に依存した動作の把握が容易になる．また，機能の追加のために状態の追加を行う際も既存の実装への影響も小さくなる．

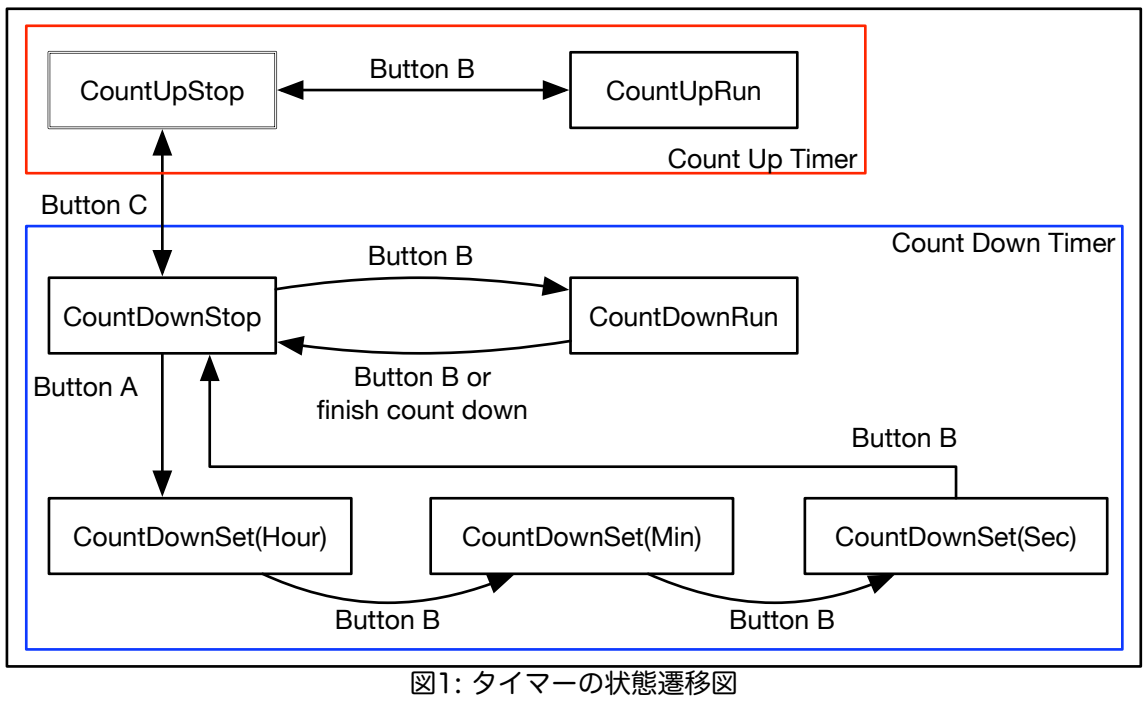

図1: タイマーの状態遷移図

```
switchmodule Timer
in   button      : ButtonPressed,
     pulse1s     : Bool
out  mode(Up)    : Mode,
     display(Display(0,0,0)) : Display
use Std, WatchIO
init CountUpStop

type DisplayPos = Hour | Min | Sec

state CountUpStop {
  node mode = Up
  …
  switch:
    if ofB(button) then
      CountUpTimerRun
    else if ofC(button) then
      CountDownTimerStop
    else
      Retain
}

state CountUpRun {
  …
}
…
```

図2：Switch拡張を用いたタイマーの記述例

```
state CountDownTimerSet(pos : DisplayPos) {
  node mode = Down

  node init[0] dh = pos of Hour -> 1, _ -> 0
  node init[0] dm = pos of Min  -> 1, _ -> 0
  node init[0] ds = pos of Sec  -> 1, _ -> 0
  …

  switch:
    if ofB(button) then
      pos of
        Hour -> CountDownSet(Min),
        Min -> CountDownSet(Sec),
        Sec -> CountDownStop
    else
      Retain
}
```

図3：カウントダウンタイマー設定状態の記述

実装

Switch拡張の処理系は純粋なEmfrpプログラムへのトランスレータとして実装されている．変換されたコードは他のEmfrpコードからサブモジュールとして利用することができる．一方で，現在の実装ではSwitch拡張を用いたモジュール記述の内部でサブモジュールを利用することができないという問題点がある．階層的な状態遷移の記述が可能になれば，状態数の多い状態遷移図におけるモジュール性を向上させることができるものと考えている．

関連研究

FRPにおいて適応的動作を表現するための既存の機構

switchオペレータ[Hudak et al. 2003]：

- 「時変値上の関数」の時変値を扱うオペレータ
- 時変値に応じて適用する関数を切り替えることで動的な動作を実現する

Emfrpに対するCOP拡張[Watanabe 2018]：

- Emfrpに文脈指向プログラミング(COP)の概念を導入したもの
- 計算の切り替えを文脈に対する条件式によって指定できる