状態遷移を表現する組込みシステム向けFRP言語の設計

松村 有倫　渡部 卓雄（東京工業大学）

概要

・状態遷移を伴った組込みシステムの開発は困難
  ・状態遷移に合わせた振る舞いの変化 + 非同期的な入力に対する応答処理
  ・これらが組み合わさったプログラムは複雑

・小規模組込みシステム向けFRP言語XStormを提案
  ・FRPによってリアクティブシステムを宣言的に記述可能
  ・状態遷移を明示的に表現できる言語機構を提供
  ・動的なメモリ管理を行うことなく振る舞いの変化を実現

関数リアクティブプログラミング(FRP)

・リアクティブシステム
  ・外部からの入力に応答して状態の更新と出力処理を繰り返すシステム
  ・例：UI，センサーを用いた制御プログラム

・関数リアクティブプログラミング (FRP)
  ・時間変化する値を時変値というオブジェクトで抽象化
  ・時変値を用いることで変化する内部状態を宣言的に表現可能

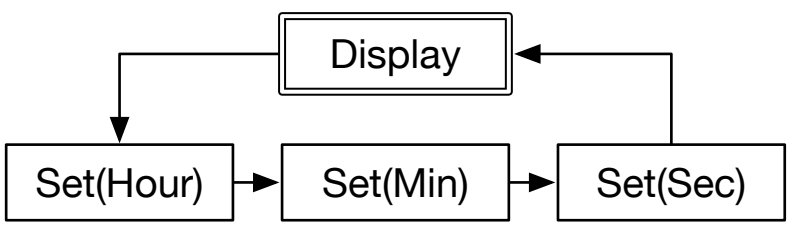
状態遷移を表現するFRP言語：XStorm

小規模組込みシステム開発に有利な特性

・XStormはEmfrp[Sawada et al. 2016]を元にしたFRP言語
  ・Emfrp：小規模組込みシステム向けFRP言語
・Emfrpが持つ以下の特性を引き継いでいる
  ・時間漏れ・空間漏れがない
  ・小さなメモリフットプリントで動作する
  ・動的なメモリ管理を行わない

状態遷移を明示的に扱う言語機構：switchモジュール

・右のコードはswitchモジュールを用いたデジタル時計の実装例
・状態ごとに出力時変値と遷移を定義
  ・状態遷移に応じて定義の切り替えを行う
・遷移可能な状態集合を定義したものに限定
  ・コンパイル時の解析を容易にして静的なメモリ管理を実現



デジタル時計の状態遷移図（二重枠は初期状態）

```
switchmodule Watch {
  in buttons : Buttons,
     curTime : Time,
  out display(Time(0,0,0)) : Time,
      curTimeUpdate : TimeEvent
  init Display

  state Display {
    …
  }

  state Set(pos : SetPos) {
    const dh = …
    const dm = …
    const ds = …

    out node display =
      if buttonB(buttons) then
        addT(display@last, dh, dm, ds)
      else Retain

    out node curTimeUpdate =
      …

    switch:
      if buttonA(buttons) then
        case pos of
          Hour -> SetMode(Min);
          Min -> SetMode(Sec);
          Sec -> DisplayMode;
      else Retain
  }
```

デジタル時計を実装したXStormコード

XStormでは時変値をノードと呼ぶ.

モジュールのヘッダで入出力となるノードと初期状態の指定を行う.

遷移する状態の定義を行う. 状態にはパラメータも設定できる.

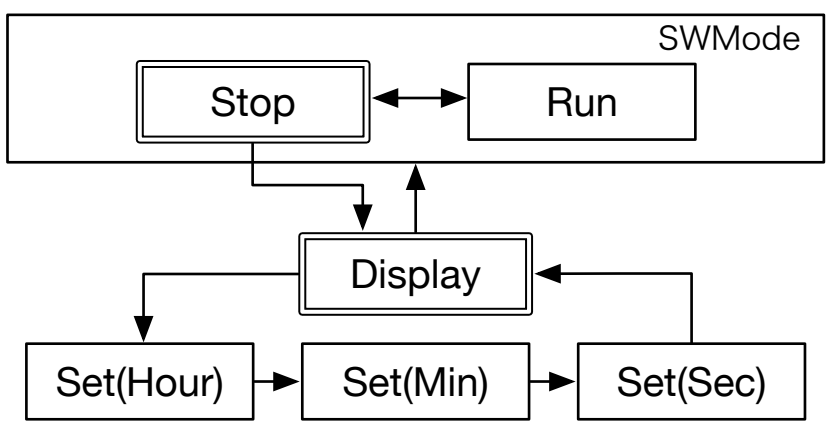
その状態における出力ノードの定義式を記述する. 定義を補助する定数やノードもここで定義できる.

次の状態への遷移を記述する.

動作の様子

状態のモジュール化

・上の例のデジタル時計を拡張する
  ・SWMode状態を追加してストップウォッチ機能を追加
・状態の追加を行っても既存の状態定義は影響を受けない
  ・コードの拡張が容易



拡張した状態遷移図

```
switchmodule Watch {
  in buttons : Buttons,
     curTime : Time,
     dt : Int #入力ノードを追加
  …
  state Display {
    … #SWModeへの遷移を追加
  }

  state Set(pos : SetPos) {
    …
  }

  #ストップウォッチモード用の状態を追加
  state SWMode {
    …
    out node timeUpdate = NoEventT

    newnode out display, exit
      = StopWatch <- buttons, dt
    …
    switch:
      if exit then Display else Retain
  }
}
```

拡張したコード

・定義したモジュールはインスタンス化して利用できる
  ・出力をノードとして参照可能

・switchモジュールを組み合わせて複雑な状態遷移を表現できる
  ・状態の階層性
  ・複数の状態遷移の独立性

```
switchmodule StopWatch {
  …
  init Stop

  state Stop {
    …
  }

  state Run {
    …
  }
}
```

ストップウォッチモジュール

共有ノード

・状態間で共有される補助ノード
・処理の一時中断などを表現できる

```
state Stop {
  …
  #Aボタンでカウントリセット
  shared node counter =
    if buttonA(buttons) then 0
    else counter@last

  out node stopWatch =
    …
}
```

```
switchmodule StopWatch {
  in …
  out …
  shared counter : Int
  init Stop

  state Stop {
    …
  }

  state Run {
    …
  }
}
```

```
state Run {
  …
  shared node counter =
    counter@last + dt

  node pulse =
    counter@last > counter

  out node stopWatch =
    if pulse then
      incT(stopWatch@last)
    else stopWatch@last
  …
}
```

関連研究

・switchコンビネータ [Hudak et al. 1997]
  ・ 関数の切り替えで時変値定義の変化を表現
・Emfrpに対するswitch拡張 [松村他 2020]
  ・Emfrpに対するswitchモジュールの導入

今後の課題

・表現力の拡充
  ・デフォルトの定義を記述する機構を提供する
  ・IO形式の切り替えを可能にする
・モデルの抽出による検証の支援