6G-03

関数型リアクティブプログラミングによる組み込みシステム向け DSL の実装

中野史彬[†1]　佐々木晃[†2]
法政大学大学院情報科学研究科[†1]　法政大学情報科学部[†2]

1. はじめに

関数型言語は，言語の特徴からバグの早期発見がしやすく，また参照透過性が維持されているためプログラムが予期しない動作を起こしにくい．そのため質の高いシステムを組むのに役立つと考えられるが，反面外部から情報を取得し，その情報に対し処理を行い値の更新をすることは不得意である．

このような処理を宣言的に記述する枠組みとして，関数型リアクティブプログラミング(FRP)が注目されている．

FRP は物理的な演算[1]やゲーム[2]，アニメーションなど時間を扱う操作や並列処理などを行う上で，非常に有用なプログラミングパラダイムである．しかし， FRP は理解するのが難しく簡単に扱えないため，実際にコーディングを行えるようになるまで時間がかかってしまう．

そこで本研究では，より FRP を扱いやすくした組み込みシステム向け DSL の実装を行う．これにより，コード量の削減や可読性，利用しやすさを向上させることができる．

2. 関数型リアクティブプログラミングとは

関数型リアクティブプログラミング(Functional Reactive Programming)とは，関数型言語の機能を維持しつつ副作用のある操作も利用できるようにしたものである．これにより，時間とともに変化する値の操作が利用しやすくなる．

例えば，各センサの値を取得しその値の和を求めるものを考える．

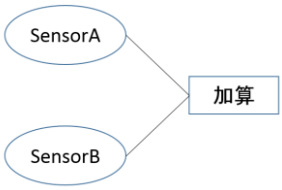

図 1　FRP での加算

FRP を利用すると各センサの値が変更されたとき，そのタイミングに合わせて値の取得を行い，その後加算処理を行い出力される値の更新が行われる．これにより，プログラマが任意のタイミングで値を取得せずに自動で更新されるため，手間を省くことができる．

FRP では，このような連続する時間に伴う変化を Behavior，離散的な時点で起こるシステムへの変化を Event という概念で表す．

また，この 2 つのことをまとめて Signal と言う．図 1 では，時間の経過が Behavior，センサの値が変化するタイミングが Event となる．FRP ではこれらの機能をもとにプログラムを記述していく．

3. FRP による組み込みシステム向け DSL の設計

本研究では， FRP による組み込みシステム向け DSL を実装する．この DSL は BrickPi と呼ばれる Raspberry Pi をロボットとして使用するためのモジュール(4 節に後述)を用いたハードウェアの制御を目的としたプログラミングを行うための言語である。

この DSL は内部 DSL に分類されるものである．機能の一例として

- BrickPi のセットアップ
- センサやモータのセットアップ
- センサやモータの制御
- 簡単な操作命令

などがあげられる．

```
updateValues :: Wire s () IO a Bool
updateValues = mkGen_ $ \_ -> do
  result <- brickPiUpdateValues
  return $ if result == 0 then Right True else if result == -1 then
Right False else Left ()
```

図 2　値の更新を行う DSL の機能

図 2 のコードは BrickPi に接続されているセンサ類が値の更新をしているかを確認し，正常に動いているかを返すことを表している．実際にコーディングするときは，この updateValues を呼び出すことにより値の更新の確認をすることができる．これを用いることにより，冗長な関数を実装せずに済むため，コード量の短縮ができ可読性の向上を行うことができる．

本研究で実装する DSL は，図 2 のようなハードウェアを用いた典型的な処理のための機能を提供し，コード量の削減と可読性の向上，利用しやすさの向上を目的としている．そのため，この DSL を利用するにあたりモータの接続箇所を固定するなどいくつかの制約がつくことがある．

4. FRP による組み込みシステムの実装

本研究では組み込みシステム向けの制御言語としての FRP を実現するためにハードウェアとして Raspberry Pi, LEGO Mindstorms，BrickPi を用いる．また FRP を利用するにあたり，汎用 FRP 向け Haskell ライブラリの Netwire を利用する．

Raspberry Pi はシングルボードコンピュータである．本研究では後述の BrickPi と接続し，BrickPi から取得した値などを Raspberry Pi で処理し制御を行う．

Implementation of DSL for embedded systems using FRP
[†1] FUMIAKI NAKANO, Graduate School of Computer and Information Sciences, Hosei University
[†2] AKIRA SASAKI, Faculty Computer and Information Sciences, Hosei University

LEGO Mindstorms は LEGO 社が提供している教育向けロボットである．本研究では，モータや各センサを利用する．

BrickPi は DEXTER Industries 社が開発したロボットキットである．BrickPi は LEGO Mindstorms のモータやセンサ類をワンタッチで接続可能にするものである．これにより，ハードウェア実装において時間がかかる工程を短縮でき，さらにハードウェア面での接続ミスなどのバグを減らすことができる．

本研究では DEXTER Industries 社が提供している BrickPi 制御用ライブラリを Haskell 処理系の ghc から利用できるようにし，その上で FRP を用いる．

Netwire は Haskell 用のライブラリで，汎用 FRP を提供する．本研究では，用いる機材の環境に対応しているバージョンが限定されているため Netwire-5.0.0 を用いる．

これらの機材を利用して FRP によるの組み込みシステム実装を行う[3].

5. 実験内容

本研究で提案する DSL の有用性を実証するにあたり，以下の 2 つシステムを実装し，それぞれコード量の削減や可読性の向上が行えているかを DSL の有無で比較する．

① ライントレースカーの実装

② 障害物を避け，目的地に停車する

①は線上を動く車両型ロボットで線の濃淡を検出し，濃淡に合わせた振る舞いをする．②は障害物を感知し，それを避けながら目的地へ向かうものである．①ではカラーセンサ，②ではカラーセンサと超音波センサを使用する．また①，②ともにモータを 2 つ使用する．

コードの一部は図 3，図 4 のようになっている．

```
car :: (HasTime t s) => Wire s () IO (Light, Light) ()
car  = proc (lightL, lightR) -> do
    _ <- motors -< (lightL, lightR)
    t <- time -< ()
    _ <- delay 50000 -< t
    _ <- updateValues -< ()
    (lightL1, lightR1) <- lineSensors -< ()
    car -< (lightL1, lightR1)
```

図 3　DSL を利用し実装した①のコードの一部

図 3 のコードは①のコードの一部である．この関数は，左右のカラーセンサからの情報を motors 関数に渡し，motors 関数内でセンサの情報に合わせた振る舞いを行う．その後センサの情報の更新を行う．

```
park :: (HasTime t s)=> Wire s () IO () ()
park = for 2 . leftMotor' --> stopLoop
```

図 4　DSL を利用し実装した②のコードの一部

図 4 のコードは②のコードの一部であり，停車するときの処理を表す．この関数は車両が停車位置についたときに呼び出され，車両がその場で一定秒間回転し，その後停車する処理を行う stopLoop 関数を呼び出す．leftMotor'は DSL で定義されたものであり，その場で左回転することを表している．

6. 考察

実装した DSL を利用し，①と②を DSL の有無でコード量の比較を行った．①と②のコード量の比較は表 1 のようになった．

	DSL 有	DSL 無
①	69 行	86 行
②	84 行	111 行

表 1　コード量の比較

関数型言語を用いて本研究で対象とするような組み込みシステムを実装するにあたりネックとなるのは関数の型の一致だが，これは DSL 側で型が厳格に定義されているため，この DSL を利用することにより大部分を型推論で通すことができる．これにより，DSL の機能を利用すると型推論が行いやすい形になっているため，型の一致を簡単にできる．

コード量の削減に関しては，本 DSL を用いることにより，まず多くの import 部分を削除することができる．開発する上で必要なライブラリ群は DSL に記述してあるため，ユーザが開発するたびに import 部分を書く手間が省かれる．

次に，DSL によって提供された部分はそのままコード量の圧縮のつながる．特に必ず呼び出す初期化の箇所やモータの簡単な動きなどが DSL で提供されているため，コード量を削減することができる．また，コード量の削減によりコード全体が見やすく，かつわかりやすくなるため可読性の向上も行われている．

可読性はアロー記法を導入しているため，入出力の流れがわかりやすいものになっている．

本 DSL を用いることにより，コード量に関しては①，②ともに 20～25%程度の削減が見られた．これは定型的に用いる箇所が DSL で定義されているため，開発者が定義する必要がなくなった点が大きく作用している．

また，この DSL のモータを制御する機能を利用するにあたってはモータの接続箇所を指定のものに固定するなどの制約があるが，この制約は同時にハードウェアに対しての制約でもある。これは，ソフト，ハードの両者を一定の型で実装することを促進するため，簡単に実装することができるようになる．

7. おわりに

本研究では関数型リアクティブプログラミングによる組み込みシステム向け DSL の実装を行った．この DSL により，目的であるコード量の削減や可読性の向上，利用しやすさの向上が図れた．

今後の課題として，この DSL で提供される環境でのテストシステムの導入があげられる．センサ類が正常に動作するか適当な値をセンサに直接入れ，正常に動作するかテストする方法や，グラフィカルなモデルを作成しシミュレーションしながらテストする方法などが考えられる．

参考文献

[1] Kenji Ohmori, “Development of Functional Re-active Programming Using an Incrementally Modular Abstraction Hierarchy”, The 5th International Conference on IT Convergence and Security 2015 in Kuala Lumpur, August 2015

[2] Antony Courtney, Henril Nilsson, John Peterson, “The Yampa Arcade”, ACM, Haskell’03, August 2003

[3] 中野史彬，大森健児，佐々木晃，“関数型リアクティブプログラミングによる組込みシステムの実現”，2016 年電子情報通信学会総合大会，D-3-1，March 2016