Vol.2019-ARC-235 No.42
Vol.2019-SLDM-187 No.42
Vol.2019-EMB-50 No.42
2019/3/18

小規模組込みシステム向け関数リアクティブプログラミング言語のためのモジュール化機構

渡部 卓雄[1,a)]　松村 有倫[1]　横山 陽彦[1]

概要：マイクロコントローラ等の小規模組込みシステム向けに設計された関数リアクティブプログラミング (FRP) 言語 Emfrp のためのモジュール化機構について述べる．具体的には，実行時文脈に依存する振る舞いの宣言的な記述を可能にする文脈指向プログラミング (COP) 機構と，状態に依存する処理のモジュール化手法，および高階関数から構成される．これらを用いることで，適応的な組込みシステムの宣言的かつ簡潔な記述および適切なモジュール化を可能にすることについて議論する．

キーワード：関数リアクティブプログラミング, 組込みシステム, 文脈指向プログラミング

1. はじめに

関数リアクティブプログラミング (Functional Reactive Programming, FRP) は，組込みシステムに代表されるリアクティブシステムの効果的な記述を支援するプログラミングパラダイムである．FRP の特徴は**時変値** (Time-Varying Value) と呼ばれる抽象化機構にある．時変値を用いて時間と共に変化する値とその関係を宣言的に記述することで，ポーリング（ループ）や割り込みハンドラ等，プログラムを分断し見通しを低下させる要因の排除が可能になる．

著者らは，小規模組込みシステムを対象とした純粋 FRP 言語 Emfrp [2] を設計・実装し，いくつかの応用を通してその有用性を明らかにしてきた．Emfrp はマイクロコントローラ等のリソースの限られた環境で実行することを前提として設計されている．具体的には静的型付けに加え，時変値を一級データではなく名前で参照することの強制や再帰の禁止といった言語設計上の制約を設けている．以上の制約により，プログラムが利用する記憶領域の大きさは静的に決定できる．加えて push 型と呼ばれる実行方式の採用により生成されるコードサイズを抑えている．これらの性質により，表現力を大幅に落とすことなく小規模組み込みシステム記述に適した言語機構を提供しているが，実行時情報にもとづく柔軟な振る舞いの表現に制約が生じる．例えば組込みシステムの実行において重要な，変化する環境への適応動作を適切にモジュール化して記述することが難しい．

このような問題に対処することを目的として，筆者らは Emfrp における自己反映計算 (reflection) の機構を提案し，ロボットの適応動作記述への応用等を通してその有効性を示した [5]．これによって適応動作の宣言的な記述が可能になったが，その記述はメタレベルモジュール内で行うため，オブジェクトレベルで記述されるアプリケーション動作とは分離される．しかし必ずしも汎用的ではない，アプリケーションに特化した適応動作を記述したい場合，アプリケーションと完全に分離した記述が必要になることで却って記述性や可読性の低下につながることがある．

現在までに筆者らは，組込みシステムに特化した言語である Emfrp への COP 機構（層と層の活性化機構）の導入が，実行時文脈に依存した処理の宣言的な記述および適切なモジュール化を可能にすること明らかにしてきた [4]．本稿では，Emfrp を組込みシステム向けのより実用的なプログラミング言語として発展させることを目的とした，言語機構に関する現在進行中のプロジェクトについて述べ，その有効性について議論する．具体的には上で述べた COP 拡張の概説に加え，状態遷移機構および高階関数機構の導入について議論する．

以下，第 2 節で FRP 言語 Emfrp について概要を，第 3 節で Emfrp への COP 拡張について述べる．第 4 節では状態に応じて実行するコードを切り替えるための COP とは異なる機構の必要性について論じる．第 5 節では Emfrp への一級関数の導入について述べ，高階関数によるモジュール化について議論する．

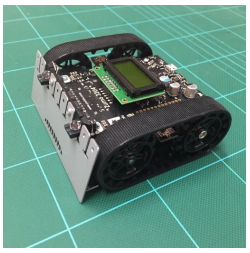
[1] 東京工業大学・情報理工学院・情報工学系
[a)] takuo@acm.org

© 2019 Information Processing Society of Japan

Vol.2019-ARC-235 No.42
Vol.2019-SLDM-187 No.42
Vol.2019-EMB-50 No.42
2019/3/18

2. 組込みシステム向け FRP 言語 Emfrp

Emfrp [2][*1]は小規模組み込みシステム向けに設計された関数リアクティブプログラミング言語である．本節では Emfrp について例を通して概要を述べる．



例として左の写真に示す小型ロボット Pololu Zumo 32U4[*2]の制御プログラム[*3]を図 1 に示す．このロボットは 2 つのモータ，ジャイロ・加速度・磁気センサ等を搭載し，マイクロコントローラ ATmega32U4（RAM2.5KB，フラッシュメモリ 32KB）によって制御される．図 1 のプログラムは，ロボットが置いてある床が z 軸（ロボットの中心を通り床に垂直な軸）を中心に回転したとき，その回転を打ち消すようにモータを制御し，結果としてロボットが常に同じ方向を向くようにする．

Emfrp のプログラムはモジュールと呼ばれる単位で構成される．図 1 のプログラムは AntiRotation という名前を持つ 1 個のモジュールからなる．モジュールのヘッダ部（1–6 行目）ではモジュール名や入出力ノードおよび使用するライブラリが宣言され，本体（8 行目以降）では型，定数，関数およびノード（後述）が定義される．

```
1  module AntiRotation
2  in  gyroZ  : Int,   # gyroscope (z-axis)
3      t(0)   : Int    # current time (usec)
4  out motorL : Int,   # left motor
5      motorR : Int    # right motor
6  use Std
7
8  # constrains the speed of the motors
9  data maxSpeed = 400
10 func motorSpeed(s) = if s < -maxSpeed then -maxSpeed
11                      else if s > maxSpeed then maxSpeed
12                      else s
13
14 # PD-control parameters
15 data kp = 11930465 / 1000
16 data kd = 8
17
18 # time difference
19 node dt = t - t@last
20
21 # calculates the angle to turn
22 node turnAngle = gyroZ * dt * 14680064 / 17578125
23
24 # calculates the turning speed
25 node speed = motorSpeed(-turnAngle / kp - gyroZ / kd)
26
27 # controls the motors
28 node (motorL, motorR) = (-speed, speed)
```

図 1 小型ロボット Zumo 32U4 の制御プログラム

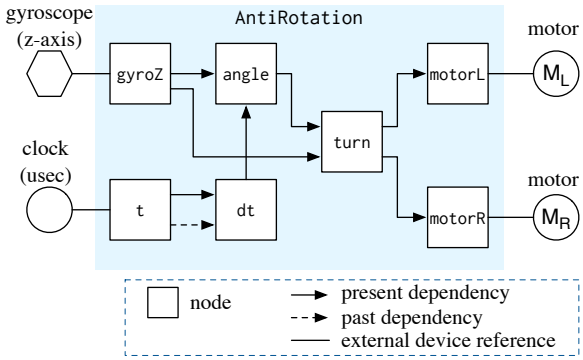


図 2 図 1 の DAG による表現

ノードは Emfrp における時変値を表す言語機構であり，その値は時間と共に変化し得る．ノードは外部機器に接続されている入力・出力ノード，およびそのような接続を持たない内部ノードの 3 種類に分類される．この例では gyroZ および t が入力ノード，motorL および motorR が出力ノードであり，それぞれジャイロセンサと時計，および左右のモータに接続されている．その他のノード（dt, turnAngle および speed）は内部ノードである．

入力ノードの値は接続されている外部機器によって決定される．この例では gyroZ および t がジャイロセンサの計測値（z 軸）および現在時刻（マイクロ秒単位）を表す．それ以外のノードの値はモジュールの本体において予約語 node を用いて定義される．図 1 では 19 行目，22 行目，25 行目および 28 行目がそれぞれ dt, turnAngle, speed および motorL, motorR の定義である．各ノードの値は=の右辺の式（ノードの定義式と呼ぶ）によって決まる．この例では，ノード turnAngle は現在の回転角を，speed はその回転を打ち消すためのモータの速度を表している．

ノード名の後に演算子 @last が付いたものは**直前値**，すなわち現在の値に至る直前（微小時間前）の値を表している．この例では微小時間を表すノード dt の定義において用いられている（19 行目）．

ノードの定義式中に他のノード名が出現するとき，前者は後者に依存するという．例えばノード turnAngle は gyroZ および dt に依存している．図 2 は図 1 におけるノード間の依存関係をグラフで表現したものである．Emfrp におけるプログラムの実行は，このグラフに沿って入力ノードから出力ノードに向かって値を更新しつづけることとして実現される．これにより，入力ノードに接続された外部機器（この例ではジャイロセンサと時計）の値の変化が出力ノードに伝搬する．そのため，ノードの依存関係はサイクルを含まない有向グラフ (DAG) となる必要がある

[*1] https://github.com/psg-titech/emfrp
[*2] https://www.pololu.com/category/170/zumo-32u4-robot
[*3] https://github.com/psg-titech/emfrp_samples

© 2019 Information Processing Society of Japan

Vol.2019-ARC-235 No.42
Vol.2019-SLDM-187 No.42
Vol.2019-EMB-50 No.42
2019/3/18

3. 文脈指向プログラミング機構の導入

適応的な動作のモジュール化を促進するプログラミングパラダイムとして**文脈指向プログラミング** (Context-Oriented Programming, COP) [1] がある．一般的な自己反映計算機構とは異なり，文脈（実行時情報）に依存した処理のモジュール化に特化した言語機構を導入することで，適応動作を見通しよく記述することを可能にする．多くの COP 言語は，Java などの既存の言語に文脈に依存する処理を分離して記述するための**層** (layer) と呼ばれる言語機構，および実行時情報に依って適切な層を活性化する実行時機構によって実現されている．現在までに，組込みシステム，特にロボットのように移動することで周囲の環境が変化するようなシステムにおける COP の有用性が示されている [3].

筆者らは，Emfrp プログラムのモジュール性改善を目指し，層 (layer) の記述を導入して COP の機能を導入するための拡張を提案した [4]．以下に当該 COP 拡張について概要を述べる．

図 3 は COP 拡張を用いて記述した Emfrp のプログラム例である．これは温度・湿度センサによるファンの制御を行うプログラムであり，不快指数が規定値以上になっている間ファンのスイッチを ON にする．この基本機能に加え，(a) 不快指数 `di` の値が閾値 `th` 近辺で変化した際にファンのスイッチが頻繁に ON/OFF を繰り返さないようにするヒステリシス制御，(b) ファンが現在までに ON になった合計時間を LCD に表示する機能，および (c) ファンのモーターを保護するため連続して 2 時間以上 ON になった際に強制的に 30 分間 OFF にする追加機能が導入されている．

各追加機能は層として定義されている．層を定義する構文は以下のように定義される．

layer L `where` e

ここで L は層の名前，e は条件式である．この定義に続くノードの定義が，定義された層が活性化されたときに有効になる．層の定義におけるノード定義に **enter**, **exit**, **retain** というキーワードが先行する場合，それぞれ層が活性化された時 (**enter**) と非活性化された時のみ更新 (**exit**) されること，および層が活性化されていないときは最後の値を保持すること (**retain**) を表している．これらのような，層な活性・非活性化に連動するノードの定義の導入により，文脈に依存する様々な振る舞いを条件式等を多用せずに簡潔に記述できるようになっている．

図 3 では，(a) は層 `HYSTERESIS`（20-22 行目）で，(b) は層 `CTIME`（25–27 行目）で，(c) は層 `OTIME` と `SHUTDOWN`（30–39 行目）でそれぞれ実現されている．それぞれの層は個別に導入や削除が可能であり，層の独立性は高い．また基本機能の記述である 10–17 行目については変更の必要がない．

```
1  module FanController2  # module name
2  in  tmp   : Float,     # temperature sensor
3      hmd   : Float,     # humidity sensor
4      clock : Int        # POSIX time system clock
5  out fan   : Bool,      # fan switch
6      ctime : Int        # LCD for time
7  use Std                # standard library
8
9  # discomfort (temperature-humidity) index
10 node di =
11     0.81 * tmp + 0.01 * hmd * (0.99 * tmp - 14.3) + 46.3
12
13 # fan switch
14 node init[False] fan = di >= th
15
16 # threshold
17 node th = 75.0
18
19 # hysteresis behavior
20 layer HYSTERESIS where fan
21   enter node th = @proceed - 0.5
22   exit node th = @proceed + 0.5
23
24 # cumulative operating time
25 layer CTIME where fan
26   enter node init[0] ctime0 = clock - ctime@last
27   retain node init[0] ctime = clock - ctime0
28
29 # continuous operating time
30 layer OTIME where fan
31   enter node init[0] otime0 = clock
32   retain init[0] node otime = clock - otime0
33
34 # stop the fan for 30 minutes
35 # after 2 hours continuous operation
36 layer SHUTDOWN where (otime > 7200 || timer > 0)
37   enter node init[0] timer0 = clock + 1800
38   retain node init[0] timer = timer0@last - clock
39   node fan = False
```

図 3 COP 拡張を用いたファン制御プログラム

COP 機構を用いずに同等の動作を記述すると，一般に実行時文脈（例えばファンが ON になっている文脈やモーター保護タイマーが有効である文脈等）に依存する動作は条件式（**if**式）を伴って記述される．しかし文脈に依存する範囲が広くなることで，同様の条件を表す式がプログラム中に散在することになる．このような状況は横断的関心事とみなすことができ，プログラムのモジュール性・可読性を低下させる．提案した COP 拡張では，文脈に依存した振る舞いの定義を層としてモジュール化することで，実行時文脈に依存した動作の宣言的かつ簡潔な記述を可能にしている．

4. 状態遷移表現のための言語機構

前節で述べた COP 拡張の応用のひとつに状態遷移図の表現がある．具体的にはレイヤーを状態に対応させ，状態遷移をレイヤー間の活性化として表現する．組込みシステムを設計する上で，システムの取り得る状態を状態遷移図

© 2019 Information Processing Society of Japan

として表現することは一般的であるが，状態をレイヤーで表現した場合，プログラム中に定義されたレイヤーが状態を表しているのかそれ以外の文脈に対応しているのかは一見して明らかではなく，プログラムの可読性に必ずしも貢献しない．

筆者の一人である松村は，FRP のための言語機構である switch [6] に相当する機構を Emfrp に導入する手法を提案している．これは図 2 におけるノード間の依存関係を実行時に動的に切り替えることに相当する機構であり，状態に応じて動作を切り替えるようなプログラムを容易に表現することを可能にする．加えて，状態遷移の表現とプログラムを明確に分離して記述することが可能になり，可読性の向上に貢献する．

5. 一級関数の導入

Emfrp の特徴の一つとして，プログラムが使用する記憶領域の大きさをコンパイル時に静的に決定できることが挙げられる．このことはプログラムの実行中にメモリを使い果たすことがないことを保証し，かつガベージコレクタが不要になることを意味する．小規模組込みシステムにおいてはこれらの性質は有利に働く．

以上の性質は，Emfrp が (1) 関数を一級データ[*4]として扱えないようにしていることと，(2) 関数やデータ構造の再帰的定義を許さないことによって保証されている．しかしこれらの条件を緩和できるか否かは明らかではなかった．例えば (1) については，一級関数をクロージャとして表現した場合，このクロージャが実行時に作られる数の上限は一般に保証されない．

筆者の一人横山は，一級関数とそれを用いる式の型に関してある条件を課すことで，プログラムが使用する記憶領域の大きさをコンパイル時に静的に決定できるという性質を保ちつつ一級関数を導入できることを明らかにした [7]．この考え方を Emfrp に適用することで，Emfrp における一級関数の利用が可能になる．つまり関数プログラミング言語で一般に用いられている高階関数によるモジュール化，例えば map や fold のような高階関数の利用や，モナド（あるいは継続渡し）を用いた例外などの扱いが可能になる．従来の Emfrp でこれらと等価な表現をするためには，一階の関数を多数定義する等の冗長な表現にならざるを得なかった．

6. まとめ

本稿では，マイクロコントローラ等の小規模組込みシステム向けに設計された関数リアクティブプログラミング (FRP) 言語 Emfrp に関する現在進行中のプロジェクトの一つであるモジュール化機構について概要を述べた．具体的には，(1) 実行時文脈に依存する振る舞いの宣言的な記述を可能にする文脈指向プログラミング (COP) 機構と，(2) 状態に依存する処理のモジュール化手法，および (3) 一級関数の導入について議論した．これらを用いることで，適応的な組込みシステムの宣言的かつ簡潔な記述および適切なモジュール化が可能になる．

[*4] 変数に代入したり，関数の引数や返値として利用できる，いわゆる普通のデータ

謝辞

本研究は JSPS 科研費 18K11236 の助成を受けている．

参考文献

[1] 紙名哲生：文脈指向プログラミングの要素技術と展望，コンピュータソフトウェア，Vol. 31, No. 1, pp. 3–13（オンライン），doi:10.11309/jssst.31.1_3 (2014).
[2] Sawada, K. and Watanabe, T.: Emfrp: A Functional Reactive Programming Language for Small-Scale Embedded Systems, *Modularity 2016 Constrained and Reactive Objects Workshop (CROW 2016)*, ACM, pp. 36–44 (online), doi:10.1145/2892664.2892670 (2016).
[3] Watanabe, H., Sugaya, M., Tanigawa, I., Ogura, N. and Hisazumi, K.: A Study of Context-Oriented Programming for Applying to Robot Development, *7th International Workshop on Context-Oriented Programming (COP 2015)*, ACM, (online), doi:10.1145/2786545.2786551 (2015).
[4] Watanabe, T.: A Simple Context-Oriented Programming Extension to an FRP Language for Small-Scale Embedded Systems, *10th International Workshop on Context-Oriented Programming (COP 2018)*, ACM, pp. 23–30 (online), doi:10.1145/3242921.3242925 (2018).
[5] Watanabe, T. and Sawada, K.: Towards Reflection in an FRP Language for Small-Scale Embedded Systems, *2nd Workshop on Live Adaptation of Software SYstems (LASSY 2017), Companion to the 1st International Conference on the Art, Science and Engineering of Programming (Programming 2017)*, ACM, pp. 10:1–10:6 (online), doi:10.1145/3079368.3079387 (2017).
[6] Winograd-Cort, D. and Hudak, P.: Settable and Non-Interfering Signal Functions for FRP: How a First-Order Switch is More Than Enough, *19th ACM SIGPLAN International Conference on Functional Programming (ICFP 2014)*, pp. 213–225 (online), doi:10.1145/2628136.2628140 (2014).
[7] 横山陽彦：組み込みシステム向け FRP 言語に対する第一級関数の導入 (2019).

© 2019 Information Processing Society of Japan