日本ソフトウェア科学会第25回プログラミングおよびプログラミング言語ワークショップ　カテゴリ3（2023年3月）

REPLをサポートする小規模組込み機器向けFRP言語処理系の実装と評価

鈴木 豪，渡部 卓雄，森口 草介（東京工業大学 情報理工学院）

関数リアクティブプログラミング（FRP）は時変値と呼ばれる時間と共に変化する値に対する関係を記述するパラダイムである．既存のFRP言語処理系はコンパイル言語が多く，プログラムの修正には手間がかかる．本研究では，小規模組込み機器における開発とプロトタイピングの支援を目的とした，REPL環境を持つFRP言語処理系Emfrp-REPLを提案し，そのインタプリタの実装をした．また，メモリ消費量とEnd-to-Endレイテンシをマイクロベンチマークによって評価し，Emfrp，MicroPythonと比較した．組込み機器の実機を用いて，Emfrp-REPLによって簡単にFRPできることをデモンストレーションする．

Emfrp-REPLの利用例

数>で始まる行はユーザの入力である．

```
1> func normalize(v, vmax) =
    if vmax <= v then vmax else v
OK, NIL
2> node tone = 523 + (normalize(tof,
    1500) * 470 / 1500)
OK, NIL
3> node tmp init[28] =
    (analog0 * 2 + tmp@last * 8) / 10
OK, NIL
4> tmp
OK, 32
5> node fan = tmp > 30
OK, NIL
6> node fan = tmp > 32
OK, NIL
```

1> 正規化のための関数を定義している．

2> 参照されている入力ノードtofは測距センサの値である．定義したノードtoneは出力ノードであり，指定した周波数の矩形波をスピーカから発音する．

3> 入力ノードanalog0（温度センサ）の移動平均を求めている．init句を用いて初期値を28にしている．

4> ノードの値をすぐに確認できる．

5> 出力ノードfanは温度が31度以上でファンが回るように定義している．

6>ファンが33度以上で回るように再定義している．

ノード

Emfrp-REPLでは，時変値はノードとして表される．ノードの定義では，他のノードを参照し，その値を用いた計算を定義できる．

入力ノードと出力ノード

入力ノードはセンサの値など，外界からの受け取る時変値である．出力ノードは計算を定義することで，そのノードの値がモータへの出力電圧など外界に影響を与える．

イテレーション

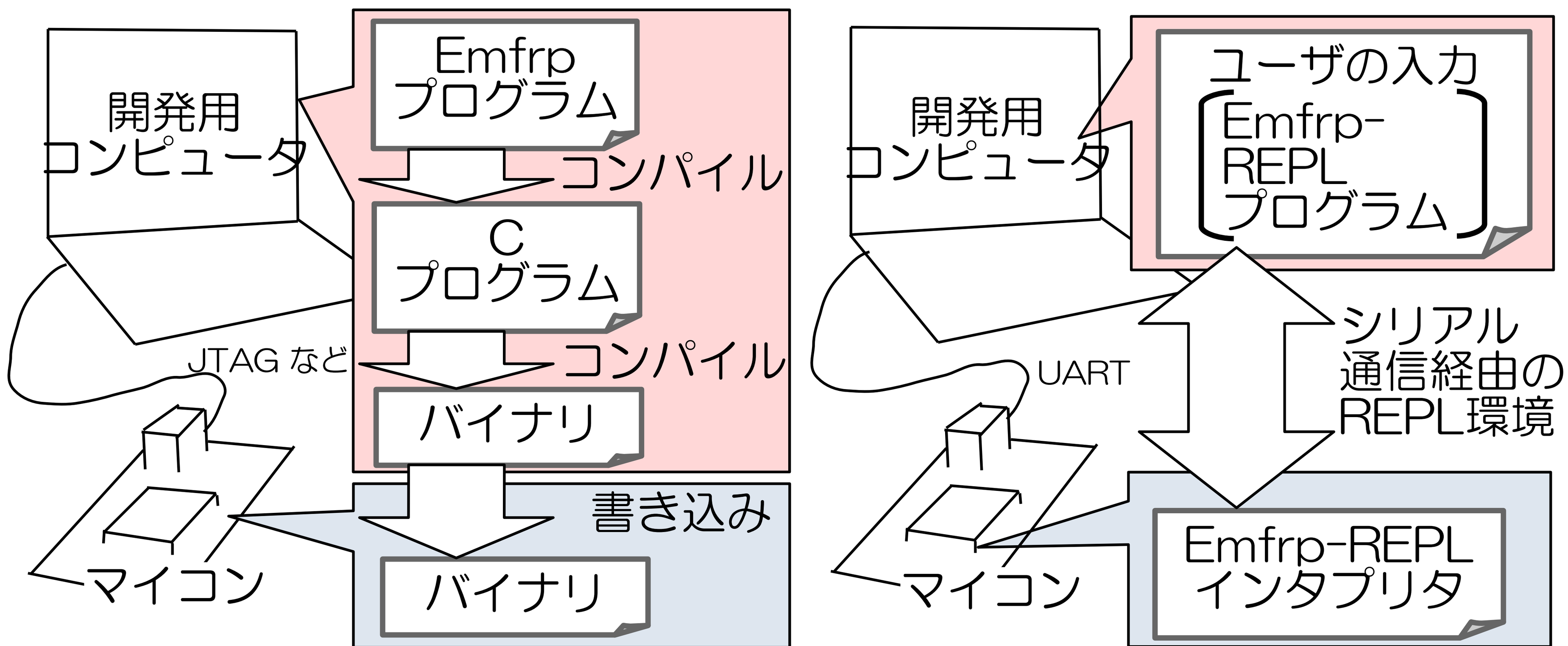
イテレーションは，ノードの参照による依存関係に従って全てのノードを評価する．

@last

ノード名に@lastを付けることで，直前のイテレーションで計算された値を参照できる．

Emfrp-REPLとEmfrp[1]の開発フローの比較

既存のFRP言語であるEmfrpでは，開発用コンピュータでコンパイルが必要であるのに対し，インタプリタであるEmfrp-REPLではREPL環境によって対話的にプログラミングできる．



実装方針

組込み開発において，次の2点は重要である．Emfrp-REPLを実装するにあたりこの2点について考慮した．

メモリ消費量

組込み機器では，記憶資源はコストと消費電力のために制限されている．プロトタイピングにおいても，大きい規模のプログラムを記述するために，メモリ消費量は重要である．Emfrp-REPL処理系を実装するにあたり，オブジェクトの表現を工夫することで，メモリ消費量を抑えた．

実時間性

実時間性のあるプログラムは，そのプログラムの実行が予測できる時間の範囲内で終わる．プロトタイピングにおいても，なるべく製品版のプログラムに挙動やパラメータを近づけるために重要な要素になる．リアクティブシステムでは周期的にイテレーションを実行することが多い．特に，積分が含まれるプログラムやバッファを持つペリフェラルを用いたプログラムにおいて，ときどき計算が次のイテレーションに間に合わないのは問題になる．

実装

プログラムの解釈

現在は，プログラムをバイトコードに変換せず，構文木をそのまま解釈する．

FRP用インタプリタの利点

クロージャを用いたFRPライブラリを既存のインタプリタ上で実現する場合にオーバヘッドが起こる．FRPでは，クロージャの外の環境で定義された変数を参照することと，関数内で変数を定義することが少ない．しかし，ナイーブなインタプリタでは，不要な環境のキャプチャや，関数内で定義された変数のためのシンボルテーブルの生成をしてしまい，メモリ消費量や処理速度が悪化する．Emfrp-REPLでは，時変値の定義をトップレベルのみに制限し，シンボルテーブルの生成を必要になるまで行わないことでメモリ消費量を抑えている．MicroPythonはバイトコードへの変換時に，変数のスコープの中を解析し，不要なキャプチャやシンボルテーブルの生成を無くす最適化をする．

オブジェクトの表現

オブジェクトの種類には，タプル（あるいはレコード），クロージャ，シンボルテーブルがある．タプルは少メモリ化のために要素数により1, 2, 3以上の3種類に区別される．オブジェクトは固定長であり，追加の領域が必要な3要素以上タプルとシンボルテーブルはごみ集め管理外に追加の領域を確保する．

ごみ集め

実時間性のために，スナップショットごみ集め[2]を実装した．このごみ集めはアロケーション時に少しずつマークかスイープを行う．マーク漏れが起こらないように参照が削除されるときにライトバリアを挿入する．

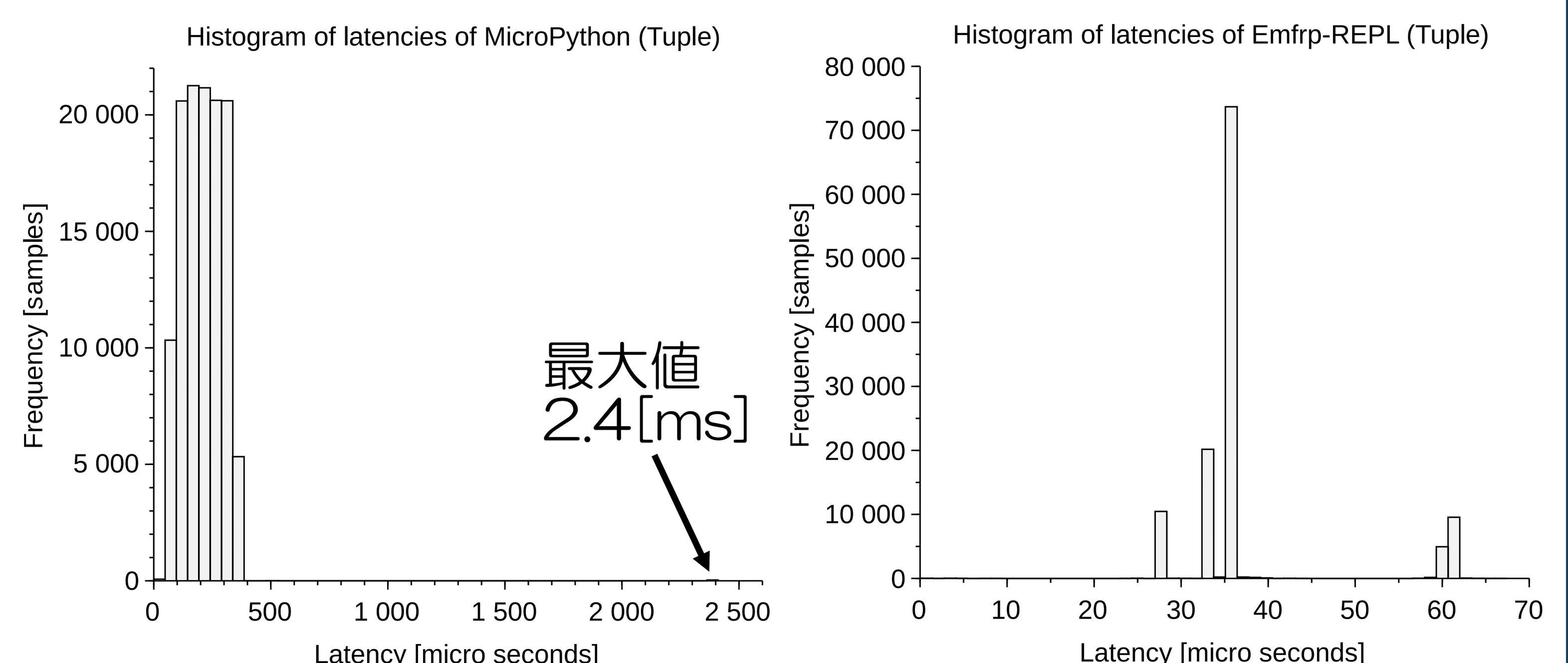
評価

```
node (v, a, b) = (gpio16, 1, 2)
node gpio17 = v
```

入力ノードgpio16から出力ノードgpio17に信号をパススルーするプログラム（右上）について，遅延とメモリ消費量を計測した．イテレーションはgpio16の割り込みによって実行される．MicroPythonにおいても，同様のプログラムで計測し，比較を行った．（ボード：ESP32-DevKitC）

メモリ消費量 いずれも16B/イテレーション程度消費した．

遅延 120000イテレーション（サンプル）計測し，下図の分布となった．ナイーブなマーク&スイープごみ集めを実装したMicroPython は2.4[ms]であり，スナップショットごみ集めの実時間性の良さがわかった．



[1] Kensuke Sawada and Takuo Watanabe. Emfrp: a functional reactive programming language for small-scale embedded systems, Companion Proceedings of the 15th International Conference on Modularity, pp.36-44, 2016.

[2] Taiichi Yuasa. Real-time garbage collection on general-purpose machines, Journal of Systems and Software, 11(3), pp. 181-198, 1990.