関数リアクティブプログラミングにおける時変値の初期化手法の提案

白鳥 佑弥　森口 草介　渡部 卓雄

関数リアクティブプログラミング（FRP）は，時間とともに変化する値を時変値として抽象化することで，組込みシステム等のリアクティブシステムを時変値の更新計算として記述するプログラミングパラダイムである．更新計算に時変値の過去の値を用いることで，変化量や累積値などのシステムの状態に依存した動作を表現できるようになるが，システム起動時には過去の値は存在しないため，あらかじめ初期値を設定する必要がある．センサの計測値や現在時刻等，実行時に決定される値について不適切な初期値を用いた場合，システム起動直後の挙動が不安定になることがある．アプリケーションごとに適当な値を想定して初期値とするといったアドホックな方法は，プログラムのモジュール化を阻害する．本研究では組込みシステム向け FRP 言語の時変値について，適切な初期値を自動的に決定する初期化手法について提案する．

1　はじめに

組込みシステムや GUI，ゲーム，ロボット制御など，外部の入力の変化に応じて内部状態や出力を変化させるシステムを**リアクティブシステム**と呼ぶ．リアクティブシステムを実現するプログラムの記述方法には，ポーリングや割込みの他に，**関数リアクティブプログラミング（FRP）**による記述が考えられる．関数リアクティブプログラミングとは，時間に応じて変化する値を時変値として抽象化し，その変化の伝播を副作用のない式として記述することでリアクティブシステム全体を表現するプログラミングパラダイムである．リアクティブシステムの表現として関数リアクティブプログラミングを採用する利点としては，1 つの値の変化がコード上に散在することがなく 1 つの式として表されること，安全性や活性の検証が容易であること [5] [6] が主に挙げられる．FRP を実現するための主な言語としては，対話式マルチメディアアニメーションのために設計された Fran [3] や，Haskell 上の DSL である Yampa [2] などが挙げられる．

本研究で対象とする **XFRP** [8] とは，組込みシステム向け FRP 言語である Emfrp [7] の後継として開発された汎用 FRP 言語である．特徴として，時間漏れや空間漏れが発生しないこと，値や関数を時変値の領域に持ち上げるための記述が必要ないこと (lift-free) が挙げられる．

XFRP では，時変値の更新直前の値（直前値）を用いて時変値の更新を行うことができる．これによって，状態に依存した動作を記述することが可能となる．ただし，システムを起動した最初の時点では時変値の更新がまだなされていないため，現状の XFRP では参照先の時変値に予め初期値として定数を設定しておく必要がある．このとき，時変値の初期値が予め決められた定数であることが原因となり，プログラムの挙動が不安定になる場合がある．例えば，センサの測定値を表す時変値に対して初期値を与えることを考える．システム起動直後のセンサの測定値は環境によって様々な値をとるが，一定時間あたりのセンサの測定値は大きく変化するとは考えにくい．一方で，定数の初期値では環境によって変化しうる測定値との差が予測できず，想定していた変化量を超えてしまう

Initialization of Time-Varying Value for Functional Reactive Programming.

Yuya Shiratori, Sosuke Moriguchi, Takuo Watanabe, 東京工業大学情報理工学院情報工学系, Dept. of Computer Science, School of Computing, Tokyo Institute of Technology.

```
1  module LPF
2  in raw: Float, t(0.0): Float
3  out filtered: Float
4  use Std
5
6  data freq = 200.0      # cutoff frequency
7  data s = 1.0 / (6.283 * freq)
8  node dt = t - t@last
9  node k = dt / (dt + s)
10 node init[0.0] filtered =
11   k * raw + (1.0 - k) * filtered@last
```

図 1　XFRP で記述された LPF のプログラム．

可能性がある．こうした想定されない時変値の変化が起こりうることで，プログラムに対して安全性を保証することが難しくなることにも繋がる．

本研究では，時変値の初期値を設定する新たな方法について提案する．初期値を定数ではなく式によって設定できるようにし，初期値を決定するために事前イテレーションという仕組みを導入する．これによって，より安定した挙動になるような初期値を各時変値に対して与えられるようになり，前述の問題を解決することができる．また，提案手法が有限時間内に終了し，リアクティブな動作に移行できることも示す．

本論文では，まず第 2 節で汎用 FRP 言語 XFRP の概要を述べ，第 3 節では具体例を用いて現状の XFRP における初期値の設定方法がもたらす問題を挙げる．第 4 節では，初期値を式によって設定する新たな初期化手法について提案する．第 5 節では，前節で提示した初期化手法によって問題が解決できること，および XFRP プログラムのリアクティブな動作に有限時間内に移行することを示す．提案手法の実際の適用例については第 6 節で議論する．第 7 節では関連研究との比較を行い，第 8 節では結論と今後の課題について述べる．

2　XFRP

XFRP [8] とは，小規模組込みシステム向け FRP 言語 Emfrp [7] に基づいて再設計された，汎用 FRP 言語である．XFRP によるプログラムの例を図 1 に示す．この例では，生の信号 `raw` と時刻 `t` を入力として受け取って，`raw` にローパスフィルタ（LPF）を適用した `filtered` を出力する．XFRP では時変値のことをノードと呼び，`node` キーワードを用いて時変値を宣言する．例えば，図中 8 行目の `node dt = t - t@last` という部分では，`dt` という識別子の時変値を宣言し，`t - t@last` という更新式に従って更新することを定義している．式中にある`@last` 演算子は，時変値の更新前の値（直前値）を表しており，これによって状態に依存した動作が可能となる．システムの起動直後は，時変値の直前値にあたる値がない．そのため，直前値を要求されるすべての時変値に対して初期値を予め設定しておく必要がある．図 1 中の時変値 `filtered` では，`init[0.0]` という部分で初期値を `0.0` に設定している．また，入力時変値にも同様に初期値を設定することができ，時変値 `t` は続く括弧内に記述されている `0.0` を初期値に設定している．

組込みシステムでは，比較的小さなメモリ環境下で，反応性の高いプログラムが要求される．そのため，時変値の過去の値を再計算することによって生じる空間漏れ（space-leak）や時間漏れ（time-leak）を防ぐ必要がある．XFRP では，時変値の過去の値として直前値しか保持しない，再帰的データ型や再帰的な関数を使わないといった制限を設けることで，空間漏れや時間漏れのないプログラムを記述できるようになっている．

XFRP の実行モデルについて説明する．XFRP では，すべての時変値の更新計算（イテレーション）を反復的に行うことでリアクティブな動作を実現している．まず，プログラム中のすべての時変値について，更新式が現在値を参照している時変値を始点，時変値自身を終点とする辺によって構成される有向グラフを事前に生成する．このとき構成される有向グラフは有向非巡回グラフ（DAG）であることを要求する．イテレーションでは，有向グラフのトポロジカルソート順に値を更新する．入力時変値については，外部から値を取得して現在値とする．それ以外の内部時変値については，更新式を評価して現在値とする．すべての時変値が更新されたら，現在のイテレーションは終

```
module TempAlert
in  tmp(20.0): Float # Temperature
out alert, d_tmp
use Std

node d_tmp = tmp - tmp@last
node alert = abs(d_tmp) > 2.0
```

図 2　温度の急激な変化を検知する XFRP プログラム．

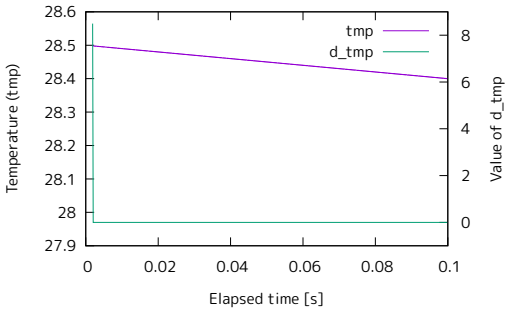

図 3　図 2 のプログラムの動作例．

了し次のイテレーションに移行する．XFRP では各時変値の参照先となる時変値が変化することはなく，イテレーションにかかる時間は概ね一定となるため，時間漏れを防ぐことができる．

3　定数初期値の問題点

XFRP では，時変値の初期値を定数で設定することになっている．しかし，初期値として適切な値がいかなる状況においても同じであるとは限らない．図 2 は，初期値が定数で設定されていることで問題が発生しうる XFRP プログラムの例を示している．このプログラムは，温度センサの値を入力時変値 tmp で受け取り，瞬間的な変化を計算して（d_tmp），変化の絶対値が 2 を超えた時点で異常として検知し出力時変値 alert を通して報告するというシステムを表している．時変値 d_tmp は時変値 tmp の直前値を参照するため，tmp には初期値を設定する必要がある．図のプログラムでは，tmp の初期値を 20.0 としている．しかし，温度センサの初期値は常に 20.0 が適切な値とは限らない．図 3 は，図 2 の動作例を示している．この例では，入力として与えられる温度センサが 20.0 よりも大きな値で始まったため，システム起動直後に大きな値の変化を検知している．しかし実際には，入力は 27.5 から 28.5 までの値を秒速 1 の速さで往復するように変化するものを与えている．そのため，どの瞬間においても入力は 2 以上の変化をせず，システムが異常を検知した場合は誤検知となる．温度センサの開始時の値はセンサが置かれている環境によって異なるため，入力時変値 tmp の初期値をどの値で設定しても，このような誤検知が発生する可能性が考えられる．

```
module DI
in tmp: Float, hmd: Float, t: Float
out di: Float
use Std

newnode tmpF = LPF(tmp, t)
newnode hmdF = LPF(hmd, t)

node di = 0.81 * tmpF +
  0.01 * hmdF * (0.99 * tmp - 14.3) + 46.3
```

図 4　LPF モジュールを適用したプログラムの例．

初期値が定数であることによる問題点は，モジュール化を考えるときにも生じる．XFRP では，newnode キーワードを用いて別モジュールの入出力として時変値を接続することができる．例として，図 1 をサブモジュールとして用いている図 4 のプログラムを考える．このプログラムでは，システムの入力として与えられる温度センサの値 tmp と湿度センサの値 hmd をそれぞれ LPF に通してから，出力である不快指数 di の値を計算するプログラムである．LPF モジュールでは，フィルタをかける前の時変値 raw の初期値は 0 であると想定し，フィルタをかけた後の時変値 filtered の初期値を 0 に設定している．しかし，tmp や hmd のようなセンサの値の初期値は環境によって変化し，またセンサそれぞれで独立した値をとる．

このように，初期値が定数であることによって，シ

```
1  module LPF
2  in raw: Float, t: Float
3  out filtered: Float
4  use Std
5
6  data freq = 200.0
7  data s = 1.0 / (6.283 * freq)
8  node dt = t - t@last
9  node k = dt / (dt + s)
10
11 init filtered = raw  # explicit init. expr.
12 node filtered =
13   k * raw + (1.0 - k) * filtered@last
```

図 5　初期化式を用いたプログラムの例.

ステムの起動直後の挙動が不安定になることがある．各時変値が初期値以外の値のみから計算される値を持っていることを表すフラグ時変値を追加することで，従来の XFRP においてもこの問題を解決することはできるが，コードが複雑になり新たなバグが発生しやすくなるほか，起動直後にしか意味を成さない時変値が残り続けることでメモリ空間を必要以上に圧迫してしまうという別の問題が発生することになる．次節以降では，初期化の方法を変えることで，コードの複雑さやメモリ使用量を抑えつつ前述の問題を解決できることを示す．

4　提案手法

XFRP の時変値に対する新たな初期化の手法を提案する．その準備として，他の時変値やその直前値の参照を含む式を初期値の代わりに設定できるようにする．このとき設定された式を**初期化式**とよぶ．図 5 は，初期化式を用いて書き直した LPF の XFRP プログラムを示している．新たに `init` 宣言を導入し（11 行目），時変値に対して初期化式を明示的に与えることが可能となっている．初期化式が明示的に定義されていない内部時変値については，更新式を初期化式とみなす．これまで定数で初期値を設定していたものは，定数を評価値とする式を初期化式に設定したものであるとみなすことができる．これにより，初期化式の導入は実質的に従来の XFRP の機能拡張となっている．

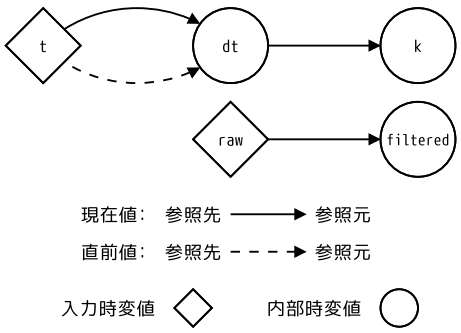

図 6　図 5 のプログラムの初期参照グラフ.

初期化の手順について説明する．まず，各時変値について，その初期化式が現在値あるいは直前値を参照している時変値を始点，時変値自身を終点とする辺を作り，有向グラフとする．このグラフを初期参照グラフとよぶ．例として，図 5 のプログラムから生成される初期参照グラフを図 6 に表す．初期参照グラフに巡回路が含まれる場合は，巡回路中の時変値の初期値を有限時間内に決定的に求めることが困難であるため，そのようなプログラムは初期化の対象外とする．初期参照グラフの生成はプログラムの静的解析で行える処理のため，コンパイル時に処理を行うことでシステム実行時の起動時間やメモリ使用量を抑えることができるほか，初期化失敗による実行時エラーを防ぐことができる．

次に，各時変値 X について，初期参照グラフにおいて X を始点としたときのパスに含まれる直前値参照の最大個数を X の遅延数 $d(X)$ とよぶ．リアクティブな動作を始める直前に，Algorithm 1 で示した手順で初期化を行い，各時変値の初期値を決定させる．各 k について，k 以上の遅延数をもつすべての時変値の暫定初期値を決定する処理を**事前イテレーション**とよぶ．図 5 のプログラムでは，`t` のみ遅延数が 1 であり，他の時変値の遅延数は 0 である．1 回目の事前イテレーションでは `t` の値をサンプリングして，その値を `t` の暫定初期値とする．2 回目の事前イテレーションでは，`t` の暫定初期値を `t@last` の値としてすべて

Algorithm 1 初期化の手順

D はすべての時変値における遅延数の最大値.
for $k \leftarrow D, \ldots, 0$ **do**
　入力時変値について，それぞれ値を取得し，取得した値を時変値の暫定初期値とする.
　for all X: 時変値 **do**
　　if $d(X) \geq k$ **then**
　　　X の初期化式を評価し，X の暫定初期値とする.
各時変値の暫定初期値を初期値とする.

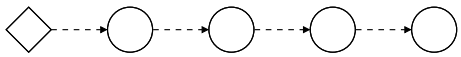

図 7　初期化にかかる時間計算量が最大となるグラフの例.

の時変値を更新し，結果を各時変値の暫定初期値とする．遅延数 0 の時変値の暫定初期値が決定したところで，暫定初期値を初期値として決定する．この例のように，事前イテレーションによって入力値がイテレーションが行われる直前にサンプリングされ初期値として設定されるため，環境に応じた適切な値が入力時変値の初期値として設定されることが期待される.

5　初期化に対する保証

この節では，4 節で述べた初期化が保証するいくつかの性質について説明する.

初期化の処理は有限時間内に終了する．ある時変値の遅延数を d とすると，初期化式に基づいて暫定初期値を求める処理を初期値が決定するまでの間に $(d+1)$ 回行う．1 つの時変値の暫定初期値の計算にかかる時間が時変値によらず一定であると仮定すると，初期化が完了するまでに各時変値はその遅延数の分だけ暫定初期値の計算を行うため，初期化にかかる時間計算量は時変値の遅延数の合計に比例する．n 個の時変値で構成された FRP プログラムにおいて，その初期参照グラフが図 7 のように直前値参照によって一列になるとき，初期化にかかる計算時間が最も長くなり，そのときの計算量は $O(n^2)$ である．有限時間内に初期化が終了し，リアクティブシステムの動作に移行することができる.

提案手法によって，システムの入力に課せられた値の変化の上限および下限に関する制約を，入力時変値においても保つことができる．ある入力 x について，時刻 t のときの値を $x(t)$，単位時間あたりの変化量の上限および下限をそれぞれ M, m とする．このとき，経過時間 Δt について，

$$m \cdot \Delta t \leq x(t + \Delta t) - x(t) \leq M \cdot \Delta t$$

という関係が成り立つ．入力 x の値を受け取る時変値 X について，最初のイテレーションにおける値を X_0，初期値を X_{init} とする．初期値を定数で設定する従来の初期化方法では，初期値の設定から最初のイテレーションまでの時間経過はほとんどないため $X_0 - X_{\text{init}} \approx 0$ であることが望ましい．しかし，X_0 が実行環境によって変化する一方で X_{init} は定数であるため，この関係は常には成り立たない．一方で，提案手法による初期化について，X に明示的な初期化式が与えられていないものとし，最初のイテレーションの開始時刻を T，最後の事前イテレーションの開始から最初のイテレーションの開始までの経過時間を τ とする．このとき，$X_0 - X_{\text{init}} = x(T) - x(T - \tau)$ であり，

$$m \cdot \tau \leq X_0 - X_{\text{init}} \leq M \cdot \tau$$

という関係を保つ．ここでは入力時変値のみを取り上げて説明したが，内部時変値についても入力値からの推論によって同様の関係が成り立っている場合は，その関係を保つことができる.

6　適用例

第 3 節で提示した問題が，提案手法によって解決されることを示す．図 8 は，図 2 のプログラムから `tmp` の初期値を取り除いたプログラムである．時変値 `tmp` は時変値 `d_tmp` によって直前値が参照されているため初期値が要求されるが，提案手法によって `tmp` の

```
1 module TempAlert
2 in  tmp: Float
3 out alert, d_tmp
4 use Std
5
6 node d_tmp = tmp - tmp@last
7 node alert = abs(d_tmp) > 2.0
```

図 8　提案手法を適用して定数初期値が不要になった例．

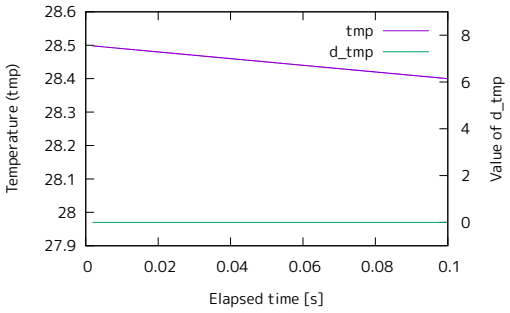

図 9　図 8 のプログラムの動作例．

値を事前に取得し自動的に初期値として設定することが可能なため，明示的な初期化式は不要である．図 3 と同じ入力を図 8 のプログラムに与えて実行した動作例を図 9 に示す．図 3 と比べて，起動直後の入力値と初期値との間で発生していた大きな変化は観測されず，異常の誤検知は発生しないことがわかる．

図 4 のプログラムについても，LPF モジュールを図 5 のような実装に変更することで接続されるそれぞれの時変値に適した初期値を事前イテレーションによって別々に設定することができる．これにより，入力時変値の初期値によらないモジュールを作成することが可能となるため，より柔軟なモジュール化を実現できるようになる．

7　関連研究

時変値の初期値として適切な値を設定するという問題は，状態に依存した動作を行うリアクティブシステムを表現する際に発生しうる一般的な問題である．状態とは過去の値の更新によって得られた値とみなすことができるが，システムの起動直後では値の更新がまだ行われていないため，起動直後の状態を初期値なしに取り出すことはできない．リアクティブシステムの動作を常に安定させるには，初期値を適切に設定する必要がある．こうしたリアクティブシステムを表現する他の FRP 言語や同期言語を例に，関連性について議論する．

Yampa [2] は，Haskell 上に実装された FRP の DSL である．Yampa では，`fby` 演算子が XFRP の初期値と同じような役割をもつため，同様の問題が発生しうる．ホスト言語上で同様の初期化処理を行ってから値を渡すことで，同じ仕組みによって初期値を設定することができる．本研究では時変値の初期化式の構造から自動的に初期化の機能を付けることができる一方で，Yampa では手動で同様の機能を付けることになるため，プログラムの変更に応じて初期化の部分も変更する必要があり，変更点が多くなってしまう．

Lustre [4] とは，リアクティブシステムを記述するための同期データフロー言語である．Lustre では，`pre` 演算子によってデータ列の直前の値を取り出すことができるが，最初に取り出される値は `nil` という未定義の値になる．Colaço らの研究 [1] では，`nil` の値がシステム全体に影響しないかどうかを型解析を用いて解析する方法について提案している．本研究では，明示的に初期化されていない時変値であっても初期化によってすべての時変値に初期値を与えることができるため，このような解析は必要ない．

8　結論

本研究では，汎用 FRP 言語である XFRP における時変値の新たな初期化手法を提案した．また，提案手法によって既存の初期値定義で生じていた問題を解決できることを示した．

提案手法によって，XFRP で記述された FRP プログラムの安全性に関する解析を容易にできるようになると期待される．第 5 節で示したように，入力に関する変化量の上限および下限に関する安全性が保証されている場合に，入力時変値や内部時変値についても同様の安全性を保証できることが提案手法によって導出できるようになる．提案手法が適用され

た XFRP プログラムの安全性の解析手法については，今後の課題とする．

謝辞 本研究の一部は JSPS 科研費 21K11822 および 19K20245 の助成を受けている．

参 考 文 献

[1] Colaço, J.-L. and Pouzet, M.: Type-based initialization analysis of a synchronous dataflow language, *International Journal on Software Tools for Technology Transfer*, Vol. 6, No. 3(2004), pp. 245–255.

[2] Courtney, A., Nilsson, H., and Peterson, J.: The Yampa Arcade, *Proceedings of the 2003 ACM SIGPLAN Workshop on Haskell*, Haskell '03, New York, NY, USA, 2003, pp. 7–18.

[3] Elliott, C. and Hudak, P.: Functional Reactive Animation, *Proceedings of the Second ACM SIGPLAN International Conference on Functional Programming*, ICFP '97, New York, NY, USA, Association for Computing Machinery, 1997, pp. 263–273.

[4] Halbwachs, N., Caspi, P., Raymond, P., and Pilaud, D.: The synchronous data flow programming language LUSTRE, *Proceedings of the IEEE*, Vol. 79, No. 9(1991), pp. 1305–1320.

[5] Jeffrey, A.: LTL Types FRP: Linear-Time Temporal Logic Propositions as Types, Proofs as Functional Reactive Programs, *Proceedings of the Sixth Workshop on Programming Languages Meets Program Verification*, PLPV '12, New York, NY, USA, Association for Computing Machinery, 2012, pp. 49–60.

[6] Jeffrey, A.: Functional Reactive Programming with Liveness Guarantees, *SIGPLAN Not.*, Vol. 48, No. 9(2013), pp. 233–244.

[7] Sawada, K. and Watanabe, T.: Emfrp: A Functional Reactive Programming Language for Small-Scale Embedded Systems, *Companion Proceedings of the 15th International Conference on Modularity*, MODULARITY Companion 2016, New York, NY, USA, Association for Computing Machinery, 2016, pp. 36–44.

[8] Shibanai, K. and Watanabe, T.: Distributed Functional Reactive Programming on Actor-Based Runtime, *Proceedings of the 8th ACM SIGPLAN International Workshop on Programming Based on Actors, Agents, and Decentralized Control*, AGERE 2018, New York, NY, USA, Association for Computing Machinery, 2018, pp. 13–22.