```
module LPF
in raw: Double, t(0.0): Double
out filtered: Double
use Std

data freq = 200.0             # cutoff frequency
data s = 1.0 / (6.283 * freq)
node dt = t - t@last
node k = dt / (dt + s)
node init[0.0] filtered = k * raw + (1.0 - k) * filtered@last
```

図 1 XFRP で記述された LPF のプログラム．

との比較を行い，第 7 節では結論と今後の課題について述べる．

2. XFRP

XFRP とは，小規模組込みシステム向けに設計された FRP 言語である Emfrp 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 では時変値はノードと呼ばれ，例えば node dt = t - t@last という部分では dt という時変値を宣言し，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．式中の@last 演算子は，時変値の更新前の値（直前値）を表している．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時変値すべて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 は続く括弧内の数字である 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，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，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，外部から値をサンプリングして現在値とする．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，イテレーションを終了する．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．

```
module AvgChange
in sensor(0.0): Double, t(0.0): Double
out change: Double
use Std

node change = (sensor -. sensor@last) /. (t -. t@last)
```

図 2 センサの平均変化率を計算する XFRP プログラム．

```
module DI
in tmp: Double, hmd: Double, t: Double
out di: Double
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 3 各センサに LPF を適用してから不快指数を計算する XFRP プログラム．

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は常に同じとは限らない．例えば，入力として与えられるセンサの値は観測された値に基づいて決定されるため，環境が変われば初期値も変化することが望ましい．他の例として，**図 2** のような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センサの値 sensor と時刻 t を受け取り，センサの平均変化率 change を出力する．時刻 t の仕様として，様々な例が考えられる．最初の更新計算時に t の値が 0 だった場合，その後の t が真に増加する値の列であったとしても，最初の出力値の計算でゼロ除算が発生し不正な値になってしまう．t が UNIX 時刻など実際の時刻に則した値である場合，最初の値は起動時に決定する．sensor の最初の値も環境に依存するため，最初の出力値は想定通りの値にならず不安定になってしまう．

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

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

図 4 初期化式を用いて書き直した LPF の XFRP プログラム．

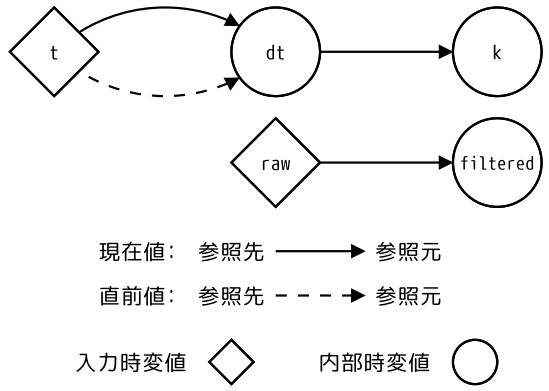

図 5 図 4 の初期参照グラフ．

いている**図 3** 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値 raw は 0 を初期値とすることを想定し，フィルタをかけた後の値 filtered の初期値を 0 としている．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．そのため，2 つのセンサ時変値の両方に同じ LPF モジュールを用いているこのプログラムでは，仮に片方のセンサの初期値が 0 で適切であったとしても，もう片方のセンサの初期値が 0 で適切であるとは限らず，結果的に起動直後のシステムの出力は変化が大きく不安定になる．

4. 提案手法

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することを許す．このとき設定された式を**初期化式**とよぶ．図 4 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．init キーワードに続けて，初期化式を定義する対象の時変値，=，そして初期化式の順に記述することで，時変値に対する初期化式の明示的な定義とする．また，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，定数を返す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ため，実質的に従来の XFRP の機能拡張となっている．

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 4 から生成される初期参照グラフを**図 5** に表す．初期参照グラフに巡回路が含まれる場合は，初期化に失敗する．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 $d(X)$ とよび，遅延数の最大値を D とする．$k = D$ として，$d(X) \geq k$ となる時変値 X について，初期参照グラフのトポロジカルソート順に次のように暫定初期値を決定する：

- 入力時変値の場合，入力をサンプリングして暫定初期値とする．
- 初期化式に時変値への参照がない定数式が設定されている場合，式の評価値を暫定初期値とする．
- 初期化式から，直前値を前回の暫定初期値，現在値を現在の暫定初期値に置き換えた式の評価値を暫定初期値とする．

暫定初期値を決定するこの処理を事前イテレーションとよぶ．$k = D - 1, \ldots, 0$ についても順に事前イテレーションを行い，暫定初期値を更新する．最終的に得られた各時変値の暫定初期値を改めて時変値の初期値として確定させる．図 4 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．よって，最初の事前イテレーションでは t をサンプリングして，その値を t の暫定初期値とする．次の事前イテレーションでは，時変値全体の暫定初期値を計算し，$k = 0$ なのでこのときの暫定初期値を初期値として確定させる．

より複雑な初期化を行う例を**図 6** に示す．この例では，入力として与えられた時変値 raw の過去 5 回の数値の中央値を出力している．初期化式のみに注目すると，各時変値には明示的な初期化式はなく，初期化式はそれぞれの更新式となっている．入力時変値である raw の遅延数は 4 であり，内部時変値 raw1, raw2, raw3, filtered の遅延数はそれぞれ 3, 2, 1, 0 である．最初の事前イテレーションでは，raw の値をサンプリングして暫定初期値とする．次の事前イテレーションでは，raw の値を再びサンプリングして暫定初期値とし，前回の raw の暫定初期値を改めて raw1 の暫定初期値とする．同様に事前イテレーションを進め，filtered の暫定初期値が決定したところですべての時変

```
module MedianFilter
in raw: Double
out filtered: Double
use Median

node raw1 = raw@last
node raw2 = raw1@last
node raw3 = raw2@last

node filtered = median5(raw, raw1, raw2, raw3, raw3@last)
```

図 6 提案手法を用いて XFRP で記述された中央値フィルタのプログラム．

値の暫定初期値を初期値として確定させる．

この方法ではプログラム上のすべての時変値の初期値を決定させたが，初期値が必要な時変値は更新式において直前値の参照先となっている時変値のみである．図 4 の dt や k のように，初期値および暫定初期値の計算が不要な時変値における暫定初期値の計算を省くことで，初期化の実行時間を改善させることができる．

5. 初期化に対する保証

この節では，提案手法による時変値の初期化が，XFRP で満たしていた性質を保つこと，初期値として適切な値を与えることについて説明する．

初期化は，有限時間内に終了する．前提として，プログラムの初期参照グラフに巡回路がないものとする．プログラム中の時変値は有限個しかないため，各時変値を始点とする初期参照グラフ上のパスは有限個の時変値を経由し，したがって時変値の遅延数は有限の値で一意に定まる．事前イテレーションでは，参照関係に従ってトポロジカルソート順に時変値の暫定初期値を計算することで，再計算することなく対象の時変値の暫定初期値を決定することができる．以上より，初期化の各手順は有限時間内に終了する処理の有限回の反復であるので，初期化は有限時間内に終了することがわかる．一般に，ノード数 n の XFRP プログラムについて，遅延数は n 以下の値になるので，初期化に必要な事前イテレーションの回数は高々 n 回である．1 回の事前イテレーションで初期値が決定されるノード数は明らかに n 以下である．このことから，ノード数 n のプログラムについて，初期化にかかる最悪時間計算量は $O(n^2)$ であることがわかる．初期化が速やかに終了することによって，XFRP プログラムが遅滞なく開始されることに繋がる．

初期化によって与えられた初期値が，定数の初期値と比較してより適切な値を与えることを説明する．まず，入力時変値については，実際の値を事前にサンプリングし，直前の値を初期値としている．このため，入力時変値の初期値は外部の環境に応じた値が柔軟に設定される．内部時変値については，初期化式を更新式によらず自由に定義できてしまうため，実行中に到達しない値を初期値に設定することが可能となってしまう．そのため，明示的な初期化式が定義された内部時変値，およびこうした時変値を初期化式で参照している時変値については，初期値が適切であるという保証はできない．明示的な初期化式が定義されている内部時変値に依存していない時変値については，実行時に取り得る入力列に基づいて実行時と同じ更新過程を経るため，実行時にも取り得る適切な初期値であることが期待できる．

初期化は，サブモジュールを接続した場合においても適切に動作する．初期化によって，プログラム中のすべての時変値に対して初期値を与えることができる．そのため．サブモジュールの入力として与えられる時変値や，サブモジュールの出力として受け取る時変値についても初期値を与えることができる．

6. 関連研究

Yampa[1] 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．

本研究では FRP 言語を対象としたが，同じくリアクティブシステムを記述する Lustre[4] などの同期言語においても似たような議論をすることができる．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 [5] 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．

7. 結論

本研究では，組込みシステム向け FRP 言語における時変値の初期値の決定方法について提案した．また，提案手法によって既存の初期値定義によって生じうる問題点を解決できることを示した．

現在，初期化式を含む XFRP プログラムに対するコンパイラ等の言語処理系の実装は未だない．そのため，提案手法によって実際に発生する起動時間やメモリ使用量のオーバーヘッドなどを計測することができない．初期化式を含

む XFRP プログラムのコンパイラの実装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] 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, pp. 7–18 (online), DOI: 10.1145/871895.871897 (2003).

[2] 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, pp. 13–22 (online), DOI: 10.1145/3281366.3281370 (2018).

[3] 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, pp. 36–44 (online), DOI: 10.1145/2892664.2892670 (2016).

[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, pp. 1305–1320 (1991).

[5] 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, pp. 245–255 (online), DOI: 10.1007/s10009-004-0160-y (2004).