# リフレクションによる時刻の扱い

鈴木 季　森口 草介　渡部 卓雄

リフレクションとは言語が実行中に自身の状態を参照・変更する仕組みであり，高い拡張性や様々な言語機能を統一的な形式で導入できるという利点がある．リフレクションでは基本的に環境や継続など言語内部の要素を扱うため，入出力や時刻といった物理的な要素を扱うものは多くない．本研究では，時間に関するリフレクションを行うことで実行時間の計測や実行の待機といった時間的操作の統一的な記述を目指す．また実際に処理系を構成する．

Reflection is a mechanism that allows a programming language to inspect and modify its own execution state at runtime. It offers advantages such as high extensibility and a unified framework for introducing various language features. Since reflection primarily manipulates internal language entities, such as environments and continuations, it rarely addresses physical entities such as input/output or time. In this study, we propose a reflective mechanism for time, aiming to provide a unified description of temporal operations, including execution-time measurement and execution suspension. We also present an implementation of the interpreter, which realizes the proposed mechanism.

## 1　導入

計算システムが自身の状態を参照・変更することをリフレクションという．またリフレクションが可能なシステムをリフレクティブシステム，リフレクティブシステムを記述するプログラミング言語をリフレクティブ言語と呼ぶ．

リフレクティブ言語はその高い拡張性から以下のような利点を持つ．

1. 言語のコア部分を小さく抑えることができる．
2. 通常アドホックな方法で導入される機能 (非正格評価，例外処理など) を統一的な形式で導入できる．
3. 対象とする問題にあった形で処理系を構築できる．

3-Lisp [12] やそれに続く Brown [7] [14]，Blond [6]，Black [1] などのリフレクティブ言語は実行中の状態として式，環境，継続を操作することができる．既存のリフレクティブ言語では実行時あるいはコンパイル時に実行系やコンパイラを構成するデータを参照・変更することが一般的であり，入出力や時刻などの外部の物理的な要素を扱うものは少ない．そこで本研究では，時間に関するリフレクションを行う言語 Timekeeper を提案し，実行時間の計測や実行の待機といった時間的操作の統一的な記述を目指す．

## 2　時刻

Timekeeper はリフレクティブ言語 Blond [6] を拡張した言語であり，実行状態として式，環境，継続に加えて時刻を持っている．Timekeeper は Blond と同様，無限個のインタプリタのタワーからなっている．評価器と時刻は全てのレベルで同一だが，環境と継続は各レベルに個別に割り当てられている (メタ継続)．そして Blond 同様，以下のリフレクティブな操作を備えている．

- `(delta (t e r k) body)`: 適用時の時刻，(未評価の) 引数，環境，継続をそれぞれ `t`，`e`，`r`，`k` に束縛

---

Handling Time through Reflection.

Minoru Suzuki, Sosuke Moriguchi, Takuo Watanabe, 東京科学大学情報理工学院, School of Computing, Institute of Science Tokyo.

- `(meaning t e r k)`: 時刻，式，環境，継続を指定してインタプリタを実行

`delta` は delta 抽象を作る操作である．delta 抽象が適用されると，その時点でのインタプリタの現在の状態が引数 `t`，`e`，`r`，`k` に束縛され，一つ上のレベルで `body` を実行する．`meaning` はインタプリタの実行状態を `(t, e, r, k)` に差し替える，つまり時刻 `t` に到達したタイミングで一つ下のレベルで `(e, r, k)` の状態から実行を開始する．ただし過去の時刻が指定された場合は即時実行する．

`time` (プログラム 1) は時刻を取得する関数である．`(time)` が呼ばれた時点での状態が `delta` の各パラメータに束縛され，継続 `k` に時刻 `t` を渡すことで `(time)` の返り値として時刻が得られる．

`time-eval` (プログラム 2) は式を取り，評価結果と実行時間を組にして返す．第一引数 `e` を `meaning` で評価し，結果を `v` に束縛する．次に評価後の時刻をプログラム 1 の `(time)` で求め (`time-eval` の呼び出しの一つ上のレベルで定義されているとする)，適用時の時刻 `t0` との差を取ることで実行時間を算出している．

プログラム 3 は `dt` 秒後に式 `e` を評価する関数を定義している．適用時の継続 `k` を実行するタイミングによって，ブロックする (`time-after-block`) かブロックしない (`time-after-nonblock`) かを制御している．`time-after-block` は単純に `dt` 秒後に `k` の下で `e` を実行することで処理をブロックする．`time-after-nonblock` は引数の式を並列実行する関数 `par` を用いている．現在の継続 `k` には適当な値 (ここでは`''()`) を渡して即座に実行してしまい，式 `e` は適当な継続 (ここでは恒等関数) の下で `dt` 秒後に実行している．

**プログラム 1　時刻の取得**

```
(define time
  (delta (t e r k) (k t)))
```

**プログラム 2　実行時間の計測**

```
(define time-eval
  (delta (t0 (e) r k)
    (meaning t0 e r (lambda (v)
    (k (cons v (- (time) t0)))))
    ))
```

**プログラム 3　実行の待機**

```
(define eval-after-block
  (delta (t (dt e) r k)
    (meaning t dt r (lambda (dt)
    (meaning (+ t dt) e r k)))
    ))

(define eval-after-nonblock
  (delta (t (dt e) r k)
    (meaning t dt r (lambda (dt)
    (par (meaning t ''() r k)
         (meaning (+ t dt) e r id)
         )))
    ))
```

## 3　並行性

並行処理の結果の利用やタイムアウトなどを表現するための機構を導入する．そのために `meaning` を拡張して，複数のインタプリタを並行に実行することを考える．複数の評価結果の扱いに応じて継続の与え方を 2 種類用意する．

- `(meaning-and t1 e1 r1 t2 e2 r2 k)`: `e1` と `e2` の両方の式が評価されるのを待ち，継続 `k` は両方の結果を取る．
- `(meaning-or t1 e1 r1 t2 e2 r2 k)`: `e1` と `e2` のいずれかの式が評価された時点でその結果を継続 `k` に渡す．もう一方の処理は中断する．

`meaning-and`, `meaning-or` は JavaScript の `Promise.all`, `Promise.any` に相当する．

`eval-concur` (プログラム 4) は 2 つの式を並行に実行する．`eval-within?` (プログラム 5) は，一定時間以内に処理が終わらなければデフォルト値 (ここでは`'()`) を返す．

## 4　実装

Timekeeper の Racket による実装を示す．

プログラム 4　並行実行

```
(define eval-concur
  (delta (t (e1 e2 f) r k)
    (meaning     t f  r (lambda (f)
    (meaning-and t e1 r
                 t e2 r
                 (lambda (v1 v2)
    (k (f v1 v2))))))
  ))
```

プログラム 5　タイムアウト

```
(define eval-within?
  (delta (t (dt e) r k)
    (meaning t dt r (lambda (dt)
    (meaning-or (+ t dt) ''() r
                t e r k)))
  ))
```

インタプリタ (プログラム 6) は以下のデータを引数に取る．

**式** `e`

**現在のレベルの環境** `r`　シンボルから値への関数

**現在のレベルの継続** `k`　結果とメタ継続を取り値を返す関数

**メタ継続** `m`　現在より上のレベルの環境と継続の組の遅延ストリーム

Timekeeper の関数は (未評価の) 引数と適用時の環境・継続，メタ継続を取る Racket 関数として実装される (プログラム 7 6 行目)．`delta` と `meaning` も Timekeeper の関数として実装され，初期環境に含められている．

delta 抽象を `(delta (t e r k) body)` により定義する (`delta` が適用されるタイミングは delta 抽象が定義されるタイミングであることに注意) と，1 行目の `a` に `((t e r k) body)` が束縛される．`delta` 抽象を実行するときは，まず引数に現在の状態を束縛し，メタ継続から一段上の環境と継続を取り出し，一段上のレベルで本体を実行する．時刻については Racket で時刻を取得しそのまま渡す (`time` は適当な基準からのシステム時間を秒単位で取得する Racket 関数 †1)．

`meaning` は，まず引数を現在のレベルで評価する．そして現在の環境と継続をメタ継続に入れ一段下のレベルで式を実行する．その際，指定された時刻と現在時刻の差分だけ sleep してから実行する．ただし差分は 0 との max を取り，過去の時刻が指定された際は即座に実行する．

## 5　関連研究

### 5.1　時刻

R2 [10] は，時間依存システムのためのリフレクティブアーキテクチャである．R2 では，リフレクティブな並行オブジェクト指向言語 ABCL/R2 [11] 上に時刻を管理する専用のメタオブジェクトを導入し，それに対するメッセージによって時間的な振る舞いの記述を実現している．一方 Timekeeper では，時刻は単に状態の一成分として扱われる．

組込みシステムやリアルタイムシステムにおける時間記述を行うアプローチとして，同期的データフロープログラミングやリアクティブプログラミングがある．同期的データフロー言語 [9] [8] では離散的なクロック信号を伴うデータの列に対する操作としてプログラムを記述する．リアクティブプログラミング言語で時間記述を取り入れたものとして [3] [4] [5] や [13] [17] がある．[3] [4] [5] は時間を時変値として与えることで時間に依存する時変値の定義が可能である．[13] [17] は時変値の更新タイミングを制御することができる．また，[2] は命令を実行するタイミングを指定できる．

### 5.2　並行性

ABCL/R [15] はオブジェクト指向並行システムにおけるリフレクティブ言語である．全てのオブジェクトにはその振る舞いを管理するメタオブジェクトが対応しており，メタオブジェクトへのメッセージによってリフレクティブな計算を実現する．オブジェクトとメタオブジェクトは並行に動作し，振る舞いの変更はメタオブジェクトのメッセージキューが消費されるま

†1　実装はプログラム 8 であり Racket 標準の `time` とは異なる．

**プログラム 6　eval**

```
(define (tk-eval e r k m)
  (cond [(const?  e) (k e m)]
        [(symbol? e) (k (r e) m)]
        [(pair?   e) (tk-eval (car e) r (lambda (f m)
                       (tk-apply f (cdr e) r k m)) m)]
        [else        (printf "invalid expression ~a\n" e)]
        ))
```

**プログラム 7　apply**

```
(define (tk-apply f a r k m)
  (match f
         [(cons 'env  f) (tk-eval (car a) r (lambda (x m)
                           (tk-eval x f k m)) m)]
         [(cons 'cont f) (tk-eval (car a) r f m)] ; jumpy
         [else           (f a r k m)]
         ))
```

で遅延される．

ABCL/R ではシステム全体の自己参照・変更はできず，オブジェクトごとのリフレクションに留まるが，ABCL/R2 [11] や ACT/R [16] は関連するオブジェクトをグループにまとめ，グループごとのリフレクションが可能である．Timekeeper では複数のインタプリタを実行できるが，`delta` は一つのインタプリタの状態しか参照できない．

## 6　今後の課題

`meaning-and`，`meaning-or` の実装や応用，Timekeeper の意味論については今後の課題である．

**謝辞** 本研究の一部は JSPS 科研費 JP24K14892 の助成を受けている．

## 参 考 文 献

[1] Asai, K., Matsuoka, S., and Yonezawa, A.: Duplication and partial evaluation: for a better understanding of reflective languages, *Lisp Symb. Comput.*, Vol. 9, No. 2–3(1996), pp. 203–241.

[2] Black, A. P., Carlsson, M., Jones, M. P., Kieburtz, R., and Nordlander, J.: Timber: A programming language for real-time embedded systems, Technical report, Technical Report CSE-02-002, Dept. of Computer Science & Engineering, Oregon Graduate Institute School of Science & Engineering, 2002.

[3] Cooper, G. H. and Krishnamurthi, S.: FrTime: Functional reactive programming in PLT Scheme, *Computer science technical report. Brown University. CS-03-20*, (2004).

[4] Cooper, G. H. and Krishnamurthi, S.: Embedding dynamic dataflow in a call-by-value language, *Proceedings of the 15th European Conference on Programming Languages and Systems*, ESOP'06, Berlin, Heidelberg, Springer-Verlag, 2006, pp. 294–308.

[5] Cooper, G. H.: *Integrating dataflow evaluation into a practical higher-order call-by-value language*, Brown University, 2008.

[6] Danvy, O. and Malmkjaer, K.: Intensions and extensions in a reflective tower, *Proceedings of the 1988 ACM Conference on LISP and Functional Programming*, LFP '88, New York, NY, USA, Association for Computing Machinery, 1988, pp. 327–341.

[7] Friedman, D. P. and Wand, M.: Reification: Reflection without metaphysics, *Proceedings of the 1984 ACM Symposium on LISP and Functional Programming*, LFP '84, New York, NY, USA, Association for Computing Machinery, 1984, pp. 348–355.

[8] Gautier, T., Le Guernic, P., and Talpin, J.-P.: Polychronous Design of Real-Time Applications with Signal, (2008).

[9] Halbwachs, N., Caspi, P., Raymond, P., and

**プログラム 8　time**

```
(define (time) (/ (current-inexact-monotonic-milliseconds) 1000))
```

**プログラム 9　delta**

```
(define (delta a r_def k_def m_def)
  (k_def (lambda (e* r_app k_app m_app)
           (let ([r1 (car (s-car m_app))]
                 [k1 (cdr (s-car m_app))]
                 [m1 (s-cdr m_app)])
                (tk-eval (cadr a)
                         (extend* r1 (car a) (list (time)
                                                   (exp-up* e*)
                                                   (env-up r_app)
                                                   (cont-up k_app)))
                         k1 m1)))
         m_def))
```

**プログラム 10　meaning**

```
(define (meaning e* r1 k1 m1)
  (match-let ([(list t e r k) e*])
             (tk-eval t r1 (lambda (t m1)
             (tk-eval e r1 (lambda (e m1)
             (tk-eval r r1 (lambda (r m1)
             (tk-eval k r1 (lambda (k m1)
             (sleep (max 0 (- t (time))))
             (tk-eval (exp-down e) (env-down r) (cont-down k)
                      (s-cons (cons r1 k1) m1))
             ) m1)) m1)) m1)) m1)))
```


Pilaud, D.: The synchronous data flow programming language LUSTRE, *Proceedings of the IEEE*, Vol. 79, No. 9(1991), pp. 1305–1320.

[10] Honda, Y. and Tokoro, M.: Reflection and time-dependent computing: Experiences with the R2 architecture, *Lisp and symbolic computation*, Vol. 9, No. 2(1996), pp. 243–277.

[11] Matsuoka, S., Watanabe, T., and Yonezawa, A.: Hybrid group reflective architecture for object-oriented concurrent reflective programming, *European Conference on Object-Oriented Programming*, Springer, 1991, pp. 231–250.

[12] Smith, B. C.: Reflection and semantics in LISP, *Proceedings of the 11th ACM SIGACT-SIGPLAN Symposium on Principles of Programming Languages*, POPL '84, New York, NY, USA, Association for Computing Machinery, 1984, pp. 23–35.

[13] Sogo, K., Tsuji, Y., Moriguchi, S., and Watanabe, T.: Periodic and Aperiodic Task Description Mechanisms in an FRP Language for Small-Scale Embedded Systems, *Proceedings of the 10th ACM SIGPLAN International Workshop on Reactive and Event-Based Languages and Systems*, REBLS 2023, New York, NY, USA, Association for Computing Machinery, 2023, pp. 43–53.

[14] Wand, M. and Friedman, D. P.: The mystery of the tower revealed: a non-reflective description of the reflective tower, *Proceedings of the 1986 ACM Conference on LISP and Functional Programming*, LFP '86, New York, NY, USA, Association for Computing Machinery, 1986, pp. 298–307.

[15] Watanabe, T. and Yonezawa, A.: Reflection in an object-oriented concurrent language, *Conference Proceedings on Object-Oriented Programming Sys-*

*tems, Languages and Applications*, OOPSLA '88, New York, NY, USA, Association for Computing Machinery, 1988, pp. 306–315.

[16] Watanabe, T. and Yonezawa, A.: An actor-based metalevel architecture for group-wide reflection, *Foundations of Object-Oriented Languages*, de Bakker, J. W., de Roever, W. P., and Rozenberg, G.(eds.), Berlin, Heidelberg, Springer Berlin Heidelberg, 1991, pp. 405–425.

[17] Yokoyama, A., Moriguchi, S., and Watanabe, T.: Switching Mechanism for Update Timing of Time-Varying Values in an FRP Language for Small-Scale Embedded Systems, *Proceedings of the 2024 13th International Conference on Software and Computer Applications*, ICSCA '24, New York, NY, USA, Association for Computing Machinery, 2024, pp. 45–54.