Vol. 27 No. 3 情報処理学会論文誌 Mar. 1986

Uranus の多重世界機構による時間推移の表現法†

中川裕志†† 中島秀之††† 柳田昌宏††

本論文では，多重世界機構を利用した世界の状態の時間的推移の表現法と推論について述べている．提案した方法は，時間情報として半順序関係を考え，時間軸を離散的時刻によって構成する．各時刻の状態変化をUranus における一つの世界とみなし，多重世界における継承 (inheritance) を用いて，時間推移による世界の状態変化を表している．また，より複雑な状態変化を扱うために，継承を選択的に打ち消すための述語も導入している．さらに可能性のある複数の時間的変化を表すための条件付き半順序関係とその遅延評価法，実時間を扱うための Time Counter についても触れている．

1. はじめに

Pat. J. Hayes は彼の高名な論文 (The Naive Physics Manifesto)[6]の中で，現代の人工知能では現実の世界から遊離した抽象的な toy world だけを対象にしているため複雑な現実的問題に対処できない，と指摘している．彼の論文はその後大きな発展を遂げた qualitative reasoning[4),7)] のバイブル的論文となった．しかしながら彼の指摘が最も顕著なのは時間概念の扱いについてであろう．planning などの分野においても時間は直接の操作対象にはならず状態空間の変化という形で implicit に扱われていたため，例えば，なにも action のない時間の推移の表現，などの問題が残されていた[2),9)]．比較的古くから知られている様相論理を時間の推移に適用した temporal logic[8]は，現在の時刻から見て別の時刻におけるいかなる assertion が見うるか(いわゆる到達可能性)についての定式化を与えてくれる．この論理は，推論時に新たな時刻間の関係をダイナミックに生成しうるものであり強力な表現力を持つ．またその拡張として temporal logic の論理式の集合をグラフとして解釈し，その上における推論をオートマトンの状態遷移として捉える方法[5]が提案されている．しかし temporal logic で表現された概念に関しての推論はできるが，temporal logic 自体，比較的粗い表現力しかなく，現実の世界における時間の流れを Hayes の言うほどに精度良く表現するのは困難である．

よりきめ細かい時間概念の表現を目指していくつかの提案がなされている．時間概念の性質に関する厳密な議論が D. McDermott の論文[9]において展開されている．彼は時間の連続性，事象 (event) やプロセスの扱い，因果関係などについて述べ，さらに未来における可能性のある世界を表す tree 状に分岐した時間軸についても言及している．ただし彼の時間論理の表現システムのインプリメンテーションについては，その困難さを述べているだけである．J. F. Allen は時間インターバルを基本表現とする時間論理について McDermott と同様の厳密な議論を展開し推論についても検討している[1)~3)]．彼は時間論理の推論システムのインプリメンテーションも進めているが，扱っている時間軸は現在注目している１本だけであり，時間の推移によって起こる可能な世界のすべてを扱う方法については述べていない．米崎[11]は区間を基礎とする時間論理に基づく論理プログラミング言語を提案している．彼のシステムも扱っている時間軸は１本であり，バックトラックによって過去の状態は変化しない．時間の推移によって可能性の生ずる世界の集合を統一的に扱おうとする試みは 10)において基本的アイデアが述べられている．

以上のように，時間論理の表現あるいはモデルに関して数々の優れた提案がなされた今日，我々は効率的なインプリメンテーションについて考える時期に来ている．可能性のある時間軸すべてを計算するナイーブな方法が計算量ないしは記憶量において組み合わせの爆発を招くことは容易に想像できる．この問題の解決は一般にかなり困難なものであろう．そこで我々はある程度的を絞って時間推移に伴う世界の状態に関する質問応答システムを念頭に置くことによって効率化を図る立場をとる．会話型システムの場合，利用者が想

† Temporal Knowledge Representation by Multiple World Mechanism of Uranus by HIROSHI NAKAGAWA (Department of Computer Engineering, Faculty of Engineering, Yokohama National University), HIDEYUKI NAKASHIMA (Electrotechnical Laboratory) and MASAHIRO YANAGIDA (Department of Computer Engineering, Faculty of Engineering, Yokohama National University).

†† 横浜国立大学工学部情報工学科
††† 電子技術総合研究所

定している時間軸は少なくとも同時には1本であろう．したがって計算機内部においても利用者の念頭にあるものと同じ時間軸が存在していれば，利用者との間での満足な質問応答ができると期待してよい．もちろん利用者の想定している時間軸は未来永劫にわたっているのではなく，ごく近い未来に関するものであることが多い．したがって，計算機内部においても比較的短いスパンの時間軸のみを実現しておき，残りの部分は利用者の要請に応じて展開する方法，いわば一種の遅延評価（Lazy Evaluation）が，計算量，記憶量の両面から望ましい方法である．本論文ではこのような観点から，時間を扱うシステム構成を提案している．

2. 多重世界記述による時間軸の表現

我々は時間軸を離散的時刻によって構成する．この際，各時刻に対応する世界の状態が存在する．注意したいのは，各々の世界はそれ以前の世界の状態に依存する点であり，ほとんどの状態は前のままで，変化するのはむしろ小さな部分である．本論文では，このような世界の状態を表現する方法として Uranus[10] の多重世界記述を用いる．ここでまず Uranus について簡単な説明をする．Uranus は Prolog/KR の多重世界機構をさらに拡張した言語であり，各々の述語定義は LISP と同じS式を用いて表現される．例えば append 述語の定義は次のように行われる．

```
(assert (append ( ) *x *x))
(assert (append (*a. *x) *y (*a. *z))
        (append *x *y *z))          (2.1)
```

ここで "*x" "*a" などの * の付いたものは変数を表している．以下，本論文では各述語定義をS式の形で表すものとする．

Uranus は動的に結合される，多くの互いに独立な世界を持ち各々の世界で定義されている述語は，外側からは見えない．ある世界に入るには WITH という述語を用いて，

```
(with 世界名  述語呼び出し)          (2.2)
```

のようにする．述語呼び出しの部分には普通の述語呼び出しのほか，定義など任意のものを任意個書くことができる．世界の名前は任意のものを使ってよく，その世界が既に存在すれば，それが使われ，存在しない場合には新たに造られる．また，世界は何重にもネストして用いることができ，その場合内側の世界からは外側の世界の定義が全部見える．この世界間のネスティングは定義時ではなく実行時に決まる．例えば世界 A, B, C で，P という述語が，(P PA), (P PB), (P PC) のように定義されているとする．これを

```
(with A (with B (with C (P *X))))   (2.3)
```

のように呼び出すと，解は

*X=PC, PB, PA

の順になる．この継承のルートは実行時に定まるので，一般にはその環境を作るために with を何重にも重ねる必要が生じる．しかし，実行時に毎回ネスティングを作るのは煩わしい．そこで，Uranus では within という述語が準備されている．within は第一引数として世界のネスティングを表すリストを取る．リストの先頭が一番内側の世界で，それから順に外側へとネストする．第2引数以降は with と同様，任意の述語呼び出しが書ける．例えば

```
(within (C B A) (P *X))             (2.4)
```

は

```
(with A
  (with B
    (with C (P *X))))               (2.5)
```

と等価である．我々は，この多重世界記述を用いて，一つの世界が時刻に対応した世界の状態を表現している，という立場をとる．さらに各世界の間に継承 (inheritance) あるいは様相論理の言葉で言えば到達可能性に関する関係が定義される．これを扱うために，within による世界間のネストを利用する．基本的には within の第一引数が時間軸上の世界の連鎖を表現していると考える．inheritance によって，ある時点での推論は，それまでの世界の状態を考慮したものになる．

3. 時間軸の遅延評価

時間に関する情報は一般的には，完全に順序付けられたものではなく，二つの時刻の間の前後関係の集合，すなわち半順序集合と考えられる．ここで，半順序関係を次のように表すものとする．

```
(→ A B)                             (3.1)
```

これは時刻Aが時刻Bより前であることを示す．なお，上記のAやBなどの半順序関係における時刻名は利用者が適当な名前を付けてよい．within を使うためには，半順序集合から，それを順番にたどることにより時間軸を構成しなくてはならない．ここで半順序関係とその時間軸との簡単な例について考えてみる．ユーザから与えられた半順序関係が次のようなもので

あるとする.

((→A B) (→B C) (→C D) (→D E)) (3.2)

この場合，得られる時間軸は (EDCBA) となる．よって(3.3)の半順序関係で世界 E における質問 P は，この時間軸のリストを直接 within の第一引数として与え，次の形で行えばよい.

(within (E D C B A) P) (3.3)

このようにして(3.2)の半順序関係に対する質問が行えるわけであるが，ここで一つ厄介な問題がある．つまり，一般的に半順序関係をたどっていくと，これより得られる時間軸は，開始時点は一意的に決まったとしても，その先において分岐し，Tree をなすことになる．within において第一引数は世界名のリストであるため，時間軸は直接的なものでなければならない．そこで，1章で述べたように注目する時間軸を1本に絞ることにする．我々は複数の時間軸が分岐する時点は外部から与えられる条件によって分岐の方向が決まると仮定する．つまり条件付き半順序関係を導入する．例えば，Aの次に起こることが条件により2通りあるような分岐点においては次のように半順序関係を表すことにする.

(IF CONDITION 1 (→A B)) (3.4)

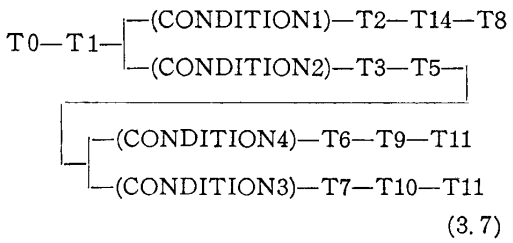
(IF CONDITION 2 (→A C)) (3.5)

これは CONDITION 1 がA の時点で真であれば (→A B) を，CONDITION 2 が真であれば (→A C) をAに関する半順序関係としてみなすという意味である．また，CONDITION の部分には述語呼び出しを書くこととし，その述語の成功，失敗により後の半順序関係が選択されることとする．この条件付き半順序関係を用いて，各分岐点において一つの条件の成立のみを許すことにすれば，一つの半順序関係のみが選ばれることになり時間軸を1本とすることができる.

この論文では以上のような半順序関係は，利用者によって完全に定義されているものとする．半順序関係が利用者によって定義されたもの以外に拡張される場合の問題点については5.2節において触れる．半順序関係の一例を次に示す.

((→T0 T1) (IF CONDITION1 (→T1 T2)) (IF CONDITION2 (→T1 T3)) (→T2 T4) (→T4 T8) (→T3 T5) (IF CONDITION4 (→T5 T6)) (IF CONDITION3 (→T5 T7)) (→T6 T9) (→T7 T10) (→T9 T11) (→T10 T55) (3.6)

この半順序関係から得られる時間軸の木は次のようになる.



(3.7)

T0-T11 は各時刻の世界の状態を表している．CONDITION1-CONDITION4 は条件を判定するための述語呼び出しであり，真偽を決めるためにユーザからの応答が必要な述語に関しては，“read” などの述語を用いてユーザへの質問を発することになる.

さて，前記のように条件付き半順序関係により1本の時間軸が得られることとなったが，実際に条件が確定していない条件付き半順序関係からどうやって時間軸を構成するかを考えてみる．これには少なくとも2通りの方法が考えられる．一つ目は，ある分岐点において条件がある場合，そこで時間軸を二つに分ける方法である．つまり，その条件について真の場合の時間軸と偽の時間軸の両方を設定しておき，後で条件が確定したとき（それまでの時間軸上での推論，ユーザへの質問の結果などによって決まる）にどちらかの時間軸を選ぶようにするのである．しかし，条件分岐が多数ある場合においては，非常に多くの時間軸の設定が必要になる．二つ目は，遅延評価を行う手法である．つまり，時間軸を構成中に条件分岐が表れた場合はそこで一時的に時間軸の構成を中止する．ユーザからの質問には，その時刻までの時間軸を用いて推論を行う．ここで，いまだに構成されていない部分についてユーザから質問が発せられた場合は構成を中止した時点の条件のチェックを行う．このとき不確定の条件の場合にはユーザへの質問を発する．そして，条件が確定した後，再び時間軸の構成を行うようにする．この二つのうち，前者はさまざまな条件の変化に対して，実行速度が早いという点では利点があるが，条件付き半順序関係の数の増加に対するメモリ量の消費を考えると実用的でない．よって，後者の “時間軸の遅延評価” を用いる.

4. 時間軸を構成する述語 construct

3章で述べた時間軸の遅延評価を実現するために，半順序関係から遅延評価により時間軸のリストを構成する述語 construct を定義する．構成された時間軸 timeline は，最も外側の世界，すなわち standard

world で assert されている大域変数である．述語 construct は次のように用いる．

construct (*t *oldline *newline) (4.1)

ここで，*t は目的の世界の名前，*oldline は今までに構成されている時間軸で，大域変数 timeline の値を表す．*newline は construct により構成される新しい時間軸であり，大域変数 timeline の新しい値として assert される．(4.1)のように construct を呼び出すと，まず *oldline の中に *t が既に存在すれば *oldline の値をそのまま *newline として返す．存在しなければ，*oldline の続きから半順序関係をたどり，*t が見つかるまで時間軸を構成する．もし，*t が見つかる前に不確定の CONDITION が見つかった場合には，そこで時間軸の構成を中止し不確定の CONDITION の実行を行う．この時に，ユーザへの質問が含まれていればユーザへの問いかけの形で条件の確定をする．条件が確定すれば，それに付随した半順序関係を用いて時間軸の構成を続行する．ここで *t が見つかればそこまでの時間軸を *newline として construct を終了する．

(バックトラック)

目的の世界が見つからない場合などに，ユーザが既存の時間軸を壊してシステムに対してバックトラックを要求したり，前に答えた条件を変更したい場合には back と入力する．back が入力されると construct により構成された時間軸を逆にたどり，一番近い条件分岐点まで戻り，それ以後の時間軸は無視する．そして，時間軸の構成を行うと再び条件のチェックを行い，別の分岐方向に向かって時間軸の構成を行う．こうして新しい時間軸が構成されると，いままで構成されていた旧時間軸上の世界は見えなくなり，バックトラックに付随して不必要なものを消し去る必要はない．

5. 時間軸上での推論

5.1 定義済みの時間軸上での場合

ここでは construct によって構成される時間軸上での述語呼び出しを含んだ質問について検討する．半順序関係は(3.6)のものを用いる．また，T0-T11 とは別の次の述語をすべての世界から見える "standard world" に定義しておく．

```
(assert (in-temp high)
        (out-temp *x) ( > *x 10))      (5.1)
(assert (in-temp high)
        (out-temp *x) (=< *x 10)
        (heater on))                   (5.2)
```

この "standard world" における(5.1)，(5.2)の定義は時間軸の構成にかかわらず，常にすべての世界から見えるものである．(5.1)の意味は，「外気の温度（定義中の変数 *x）が10度よりも高ければ室内の温度は暖かい」，(5.2)は「外気の温度が10度よりも低く，ヒータのスイッチが入っていれば室内の温度は暖かい」，ことを意味している．次に，T0, T1 に以下の述語を定義する．

```
(with T0 (assert (out-temp 5)))                    (5.3)
(with T1 (assert (heater *cond) (read *x)
          (if (== *x on)
           (with T1
            (and
             (asserta (heater off) !
                      (fail)))
             (asserta ((heater on) ! ))))
          (with T1
            (and
             (asserta ((heater on) !
                      (fail)))
             (asserta ((heater off) ! ))))
          (== *cond *x))))                         (5.4)
```

(5.3)は T0 時点で外気温が5度であったことを示している．(5.4)は T1 でユーザに対してヒータの状態を聞くための述語である．また，(3.6)の CONDITION1 の実際の述語呼び出しは次のものとする．

(heater on) (5.5)

CONDITION2 は，

(heater off) (5.6)

とする．

さて，このとき，時刻 T2 において，(in-temp high) を次の述語 at によって質問する．

(at T2 (in-temp high)) (5.7)

at は次のように定義されている．

```
(assert (at *t *p)
        (construct *t *oldline *newline)
        (within *newline *p))          (5.8)
```

at の第一引数は質問をする時刻の世界名を，第二引数はその時点で質問したい述語呼び出しである．なお旧時間軸 *oldline，新時間軸 *newline は，各々大域変数 timeline にセットされていたもの及び新たにセットされるものである．at を実行すると(5.7)の

場合，まず construct によって T2 までの時間軸の構成を試み，これに成功すればその軸上において (heater on) を実行する．T0 から出発した場合，(5.7)の質問では (heater on) が不確実であるので，システムはこれにマッチする(5.4)を実行する．(5.4) はユーザへの heater の状態の問い掛けを行うので，ユーザが "on" と入力すれば，T1 に (heater on)！が assert され，(5.5)つまり CONDITION1 が成功する．そして，新しい時間軸 *newline は (T2 T1 T0) となる．したがって，(5.7)の質問は結局 (within (T2 T1 T0) (in-temp high)) と変換され，(5.2)とマッチして成功する．しかし，(5.4)の質問に "off" と答えると T1 に (heater off)！が assert され，CONDITION1 が失敗するため，CONSTRUCT は，CONDITION2 つまり (heater off) を実行する．すると，(5.4)によって (heater off)！が既に assert されているため CONDITION2 は成功する．よって時間軸は (……T5 T3 T1 T0) と構成されていく．しかし定義されている半順序関係から構成される時間軸は有限で尽きるため，T2 が見つからずに (in-temp high) は失敗する．このような方法によって，時間軸はユーザの質問によってそれ以前の世界で assert されている assertion 及びユーザ自身の応答によって妥当な1本が遅延評価的に構成されていく．もちろんユーザがこれ以外の時間軸をチェックすることを望めば，back によってバックトラックをシステムに要求して，時間軸を遡る処理が行われる．この場合，(5.4) で asserta した (heater on), (heater off) の後ろの！の効果で，同じ質問を利用者に重複して問うことはない．

5.2 未定義な時間軸上での推論における問題点

前節で述べた推論方法はあらかじめ定められた半順序集合においては問題はない．しかし，temporal logic で見られるように，推論の実行に際してもともと与えられていた半順序関係から構成される以外の時間軸を新たに生成しその上での推論を行おうとすると，以下に述べる問題点が生ずる．

例えば(5.7)の質問に答えるための推論において (heater off) が入力されて CONDITION2 が成立した場合について考えてみる．このとき，与えられた半順序集合から構成される以外の時間軸についても質問の成功する可能性を尽くさなければならないとすると，前節で述べたように入力パラメータとして与えられた時点 T2 が見つからないという理由で質問に失敗する方法は使えない．なぜなら未来のいかなる時刻においても質問が失敗することを証明する必要があるからである．このような証明は時間に関する様相オペレータ，及び直後の時刻の概念 (いわゆる next time operator) を備えた temporal logic[8] の枠組みにおいて定式化されている．したがって，上記の問題は本システムに様相オペレータを導入すれば可能である．しかし多重世界の構成及び各世界における状態の両者にわたる推論が必要なため，推論システムに関するより詳細な検討が要求され今後の課題と考えられる．

6. 継承の打ち消しの導入

"時刻の経過に対する店の商品" を考えてみよう．時刻 T0 で品物Aが入荷し，時刻 T1 を経て，時刻 T2 で売り切れてしまった，という時間的推移を表現する方法について考えてみる．T0 で商品Aが入荷して売られている状態は

(with T0 (assert (sale A))) (6.1)

だが，T2 で売り切れたことは，どう表現したらよいだろうか？，まず次のように (sale A) を retract することを思いつく．

(with T2 (retract (sale A))) (6.2)

しかし，この方法では T0 の (sale A) が消えてしまい，例えば，T1 へバックトラックした際にもはや (sale A) が成り立たなくなってしまい都合が悪い．そこで (sale A) の継承だけを選択的に打ち消すことが必要になる．そのために hide という述語を導入した．すなわち，上記の状態は

(with T2 (hide (sale A))) (6.3)

と書ける．これにより T2 以後は (sale A) はもはや見えなくなる．しかし，T0 の (sale A) は retract されてはいないので，T1 にバックトラックしてくると (sale A) は見える．なお，hide では打ち消すのは，(sale A) だけである．したがって，もし商品Bも売っていた，すなわち (sale B) も assert されていたとしても，(6.3)は (sale B) には影響を及ぼさない (この点が Uranus の define と異なる)．assert と hide の使い分けによって，時間の推移に応じて世界に付け加えられるもの，除去されるもの，をスムーズに表現できるようになった．

7. Time Counter

現実の世界における時間を表現するためには，上記の半順序関係に加えて実時間を扱うこと，つまり単位

時間を定めての時刻カウント機能が要求される．このために我々は二つの述語 when と forward を導入する．

```
(assert (when *timeline *tnow *nowtime
        *d *tf *ftime *c *p *q)
    (within *timeline
      (if (< *nowtime *ftime)
        (if *c *p
        (forward *timeline *tnow
        *nowtime *d *tf *ftime *c
        *p *q))
        *q)))                          (7.1)

(assert (forward *line *tnow *nowtime
        *d *tf *ftime *c *p *q)
    (+ *nowtime *d *newtime)
    (newname *tnow *tnext)
    (retract (→ *tnow *tf))
    (assert ( → *tnow *tnext))
    (assert ( → *tnext *tf))
    (construct *tnext *line *newline)
  (when *newline *tnext *newtime
    *d *tf *ftime *c *p *q))
                                       (7.2)
```

when は，*tnow から未来の時刻 *tf までの間に条件 *c が成立すれば *p，成立しなければ *tf において *q を実行する，を意味する．なお *tnow から *tf に至る関係 (→ *tnow *tf) はあらかじめ定義されていたとする．*nowtime は *tnow に対応する実時刻を表す数値である．*ftime は *nowtime を起点にして測った，*ft に対応する実時刻である．forward はまず実時刻を *nowtime から *d 単位に進め，新たな実時刻 *newtime を計算する．次に *newtime に対応する時刻名 *tnext を作る．述語 newname は第一引数の時刻名に適当な一文字を加えて新たな時刻名を作る機能を持つ．そして *tnow，*tf 間の半順序関係を *tnext を含む形，すなわち (→ *tnow *tnext)，(→ *tnext *tf) に更新する．この時点で時間軸 *line は (*tnow…) のように *tnow が最新の時刻になっている．そこで construct により時間軸は *tnow の次の時刻すなわち *tnext を含む形 (*tnext *tnow…) に再構成された *newline となる．この *newline を新たな時間軸として，再帰的に when を呼び出す．我々は when を用いて実時間に関する時間論理を表現し推論することができる．例えば，*tnow から5秒後までに ACK 信号が受信されれば *nextmessage を送信し，そうでなければ *oldmessage を (再) 送信する，という通信プロトコルにおけるタイムアウト処理は次の述語 timeout で表現される．

```
(assert (timeout *oldmessage *nextmessage
        *tnow *timeline *tlimit)
        (when *timeline *tnow 0 1 *tlimit 5
        (receive ACK) (send *nextmessage)
    (send *oldmessage))                (7.3)
```

ここで，この timeout を用いた実例を示す．ただし，(receive ACK) はユーザに対して聞くものとし，(send…) は DISPLAY へ出力するものとする．ここで，現時刻を T1 (その実時刻=0)，時間軸を (T1)，timeout の発生する未来の時刻を T2 (その実時刻=5)，

```
(timeout mes0 mes1 T1 (T1) T2)         (7.4)
```

の述語で timeout 処理を表現する．これを実行すると (receive ACK) により，

ACK received?

と問い掛けてくる．
これに，5回 “no” と答えれば，時間切れとなり，mes0 が返ってくる．しかし，5回以内に “yes” と答えれば，mes1 が返ってくることになる．

when を続けて使ったときに，一つ目の when の終点実時刻 ftime を次の when の起点実時刻 *nowtime に設定すれば，長期間に渡る実時間の推移も表現できる．

8. おわりに

本論文では，多重世界記述を利用した時間論理の表現と推論について検討した．提案した方法は多重世界における継承 (inheritance) により時間推移による世界の状態の変化に対応している．このとき，打ち消し assertion の導入の必要性についても言及した．さらに複数の可能な世界を扱うに際しては条件付き半順序関係を利用している．また Time Counter によって実時間を扱う方法についても考察している．ただし本システムは半順序関係がすべて定義されている場合のみを扱っており，その応用は各種マニュアル等に現れる時間推移を含む手続きや操作系列に関する知識を知識ベース化したシステムにおける知識表現及び推論が挙げられる．今後は，5.2 節で述べた様相オペレータの導入，よりきめの細かい時間論理の表現と推論法，

効率的なインプリメンテーション並びに各種分野への応用について検討していきたい.

謝辞 斉藤康己氏をはじめとする人工知能勉強会 AIUEO の諸氏との有益な討論に感謝いたします.

参考文献

1) Allen, J. F.: Maintaining Knowledge about Temporal Intervals, *CACM*, Vol. 26, No. 11, pp. 832-843 (1983).
2) Allen, J. F.: Towards a General Theory of Action and Time, *Artif. Intell.*, Vol. 23, pp. 123-154 (1984).
3) Allen, J. F. and Koomen, J. A.: Planning Using a Temporal World Model, 8th IJCAI, pp. 741-746.
4) Forbus, K. D.: Qualitative Process Theory, Artificial Intelligence Laboratory, A. I. Memo-664, Cambridge: M. I. T. (1982).
5) Fusaoka, A. et al.: A Description and Reasoning of Plant Controllers in Temporal Logic, 8th IJCAI, pp. 405-408 (1983).
6) Hayes, P. J.: The Naive Physics Manifesto, in Michie, D. (ed.), *Expert Systems in the Micro-Electronic Age*, Edinburgh University Press, Edinburgh (May 1979).
7) de Kleer, J. and Brown, J. S.: The Origin, Form and Logic of Qualitative Physical Laws, 8th IJCAI, pp. 1158-1168 (1983).
8) Manna, Z.: Verification of Concurrent Programs, Part 1: Temporal Framework, STAN-CS-81-836 (1981).
9) McDermott, D.: A Temporal Logic for Reasoning about Process and Plans, *Cog. Sci.*, Vol. 6, pp. 101-156 (1982).
10) 中島秀之他: Prolog/KR から Uranus へ—多重世界機構の拡張—, 情報処理学会知識工学と人工知能研究会, 36-2 (1984).
11) 米崎他: 時間論理プログラミング言語 Templog, 日本ソフトウェア科学第1回大会, IE-4 (1984).

(昭和60年3月19日受付)
(昭和60年10月17日採録)

中川 裕志

昭和28年生. 昭和50年, 東京大学工学部電気工学科卒業. 昭和55年, 東京大学大学院博士課程修了. 工学博士. 昭和55年4月より横浜国立大学工学部勤務. 現在, 同大工学部電子情報工学科助教授. 人工知能の研究に従事. AIUEO, 電子通信学会, 日本認知科学会, 日本ソフトウェア科学会, 各会員.

中島 秀之

昭和27年生. 昭和52年東京大学工学部計数工学科卒業. 昭和53～54年 MIT 留学. 昭和58年東京大学大学院情報工学専門課程修了. 工学博士. 現在, 電総研人間機械システム研究室. 人工知能に興味を持つ. 著書「Prolog」,「知識表現と Prolog/KR」. AIUEO, 日本認知科学会, 日本ソフトウェア科学会, ACM 各会員.

柳田 昌宏

昭和36年生. 昭和60年, 横浜国立大学工学部電気工学科卒業. 現在, 同大大学院在学中. 人工知能の研究に従事.