（１９８５．３．１９）

時区間を基礎とする時間論理プログラミング

東京工業大学工学部情報工学科
米崎 直樹・新 淳・蓬莱 尚幸

１．まえがき

様々の知識表現、例えば、自然言語の意味処理やプロセスのモデル化等において、時間に関する知識を表現し、それらについて論証する事の重要性が認識されている。[1]～[6],[8] さまざまの試みが成されてきているが、プログラミング言語としてその考え方を取り込んだものはほとんど存在しない。[7]

我々は知識表現の為の新しい枠組み、Temploｇ と呼ばれるプログラミング言語を提案した。[9] その名前が示唆している(TEMporal reasoning+Programming+LOGic)ように、これは一階の述語論理に様相論理を組み込んだものである。

本稿では、まず時間に関する知識、特に履歴を表現する際に必要な条件を考察し、時区間の概念を導入する。次に、一階の区間を基にした時間論理を定義した上で、それに基づく論理型言語Templog の詳細を述べる。

さらに、時間概念を構造化する為には、より高度の抽象化が必要になると考えられるが、Templog では、区間に関してオブジェクト指向的な考え方を導入することによってこれを実現している。

２．時間の取り扱い方について

２．１． 時間に関する知識を表現する際の条件

時間に関する知識を表現する際には、次のものが最低限必要であると考えられる。

（１）時間に関する知識は相対的なものであり、また、全順序的ではない。

我々の時間に関する知識は、絶対時間とはあまり関係がなく、他のイベントに対する相対的な関係としてとらえる方が多い。例えば、

「仕事を終えてから帰ってきた。」
「帰る前に友人に会った。」

という記述を行なうのが普通であり、またこの場合は「仕事が終わった」のが「友人に会った」前か後かは特定していない。

（２）時間に関する知識は、不確定性が高い。

「夏休みに海に行った。」

と言った場合に、「夏休み」というイベントのどの時点で海に行ったのかは、曖昧である。

（３）時間の単位は、可変であるべきである。

取り組んでいる問題の要求に応じて、意識する時間の単位は異なる。例えば、歴史について論証しようとしている人にとっては、この単位が１日であれば十分であると考えられるが、論理回路の設計者にとっては、これは永遠以外の何者でもない。また、問題によっては、問題のさまざまの段階によってこの単位が変わることがありうる。

（４）持続性の概念を持つべきである。

「今朝、鍵を机の上に置いた。」

という事実を知った後は、「鍵が取り去られる」という事実を知るまでは、ずっとそこにあることを仮定することが自然である。

２．１．１． 時区間としてのイベント

我々は、イベントに関する次の前提を設ける。

「すべてのイベントは時区間である。」

ここで、イベントは、瞬間的であって、これを区間として捕えることは不適であるのではないか、という疑問があるかもしれない。例えば、

「部屋に入る。」

というイベントは瞬間的であると考えられる。

しかし、このように瞬間的であると考えられるイベントでも、これをさらに一連のイベントの列として記述することが可能である。

「ドアの前に行く。」
「ドアを開ける。」
「部屋に足を踏み入れる。」

更に、「ドアを開ける」というイベントも次のようなイベントの列に分解することが可能である。

「ドアのノブに手をやる。」
「ノブをまわす。」
「ドアを押す。」

つまり、あるイベントに関して、その時区間としてのイベントの詳細に関心がなければ、これを瞬間的な時点としてのイベントとみなすことが可能であるといえる。したがって、すべてのイベントを統一的に、幅の無いものを含めて時区間として捕えることが自然である。

３．一階の区間を基にした時間論理

以下にこの様な考え方にもとずく、一階の区間を基にした時間論理を示す。

３．１．Syntax

1) 基本述語

基本述語の構文は$P(t_1,\ldots,t_n)$で与えられる。
ここで、

P : 命題(n=0) シンボル、
あるいは述語シンボル(n>0)
t_i $(0 \leq i \leq n)$: 項

但し、項は通常の様に、変数、定数および関数記号によって構成される。

2) 式

2-1) すべての基本述語は、式である。

2-2) A、BおよびGを式とし、xを変数とすると、以下のものは式である。

$\sim A$
$A \wedge B$
$\forall x\ A$
$<A \rightarrow B>G$
$<A \overset{①}{\rightarrow} B>G$
$<A \leftarrow B>G$
$<A \overset{①}{\leftarrow} B>G$
$\Diamond G$

また、非様相述語というクラスを定義する。これは、Semantics の定義に使用する。

非様相述語：

1) すべての基本述語は非様相述語である。
2) A、Bを非様相述語とし、xを変数とすると、以下のものは非様相述語である。
   $\sim A$
   $A \wedge B$
   $\forall x\ A$

3．2．Semantics

時刻 t_i は定数に対する値の割り当てである。詳細は一般的定義と同様であるので、ここでは省略する。

時刻の列$<t_0,\ldots,t_n>$ に対する式の真偽値は、以下のように定義される。但し、$\mathbf{V}_{<t_0,\ldots,t_n>}$ Aによって式Aの区間$<t_0,\ldots,t_n>$ における真偽値を表わす。

まず、Aが非様相述語であれば、

$$\mathbf{V}_{<t_0,\ldots,t_n>} A = \mathbf{V}_{t_n} A$$

である。但し、$\mathbf{V}_{t_m} A$は、時刻t_mによって、通常のように解釈されたAの値を示す。

Aが様相記号を持った式であれば、その真偽値は次のように決定される。

$$\mathbf{V}_{<t_0,\ldots,t_n>} <A \rightarrow B>G =$$
$$\exists j\,[0 \leq j \leq n,\ \mathbf{V}_{<t_0,\ldots,t_j>} A,$$
$$[0<j \rightarrow \mathbf{V}_{<t_0,\ldots,t_{j-1}>} \sim A],$$
$$\exists l\,[j \leq l \leq n,$$
$$\forall m[j \leq m \leq l \rightarrow \mathbf{V}_{<t_0,\ldots,t_m>} \sim B],$$
$$[l<n \rightarrow \mathbf{V}_{<t_0,\ldots,t_{l+1}>} B],$$
$$\mathbf{V}_{<t_j,\ldots,t_l>} G]]$$

$$\mathbf{V}_{<t_0,\ldots,t_n>} <A \overset{①}{\rightarrow} B>G =$$
$$\exists j\,[0 \leq j \leq n,\ \mathbf{V}_{<t_0,\ldots,t_j>} A,$$
$$\forall k[0 \leq k<j \rightarrow \mathbf{V}_{<t_0,\ldots,t_k>} \sim A]$$
$$\exists l\,[j \leq l \leq n,$$
$$\forall m[j \leq m \leq l \rightarrow \mathbf{V}_{<t_0,\ldots,t_m>} \sim B],$$
$$[l<n \rightarrow \mathbf{V}_{<t_0,\ldots,t_{l+1}>} B],$$
$$\mathbf{V}_{<t_j,\ldots,t_l>} G]]$$

$$\mathbf{V}_{<t_0,\ldots,t_n>} \Diamond G =$$
$$\exists i\,[0 \leq i \leq n \rightarrow \mathbf{V}_{<t_0,\ldots,t_i>} G]$$

ここで、$<A \leftarrow B>G$および$<A \overset{①}{\leftarrow} B>G$の意味は、それぞれ右矢印の場合と同様に定義されるが、この場合は、時刻を過去から現在にとって考え、まずBが成立する時刻を考える。

また、

$$[A \rightarrow B]G \equiv \sim<A \rightarrow B>\sim G$$
$$[A \leftarrow B]G \equiv \sim<A \leftarrow B>\sim G$$
$$\Box G \equiv \sim\Diamond\sim G$$

である。また、$\supset$，$\equiv$，$\exists$は通常のように定義される。

各論理式の直感的な意味は、次の通りである。

$<A \rightarrow B>G$

現在評価中の区間において、Aが成立し始めた時刻から将来Bが初めて成立する直前の時刻よりなるある部分区間でGが成立することを表わしている。

$[A \rightarrow B]G$

従って、これは評価中の区間において、Aが成立し始めた時刻から、将来Bが始めて成立する直前の時刻よりなるすべての区間で、Gが成立することを表わす。

$<A \overset{①}{\rightarrow} B>G$

これは、評価中の区間において、初めてAが成立し始めた時刻から、将来Bが初めて成立する直前の時刻よりなる区間で、Gが成立することを表わす。

これからわかるように、様相オペレータ$<A \rightarrow B>$と$[A \rightarrow B]$との関係（および$<A \leftarrow B>$と$[A \leftarrow B]$）は、$\Diamond$と$\Box$との関係に似ている。

次に、特別の命題、START とEND とを定義する。

$\mathbf{V}_{<t_0,\ldots,t_n>} <$START $\cdots\cdots>$ においては、
$\mathbf{V}_{t_0}$START = true
$\mathbf{V}_{t_i}$START = false $(1 \leq i \leq n)$

$\mathbf{V}_{<t_0,\ldots,t_n>} <\cdots\cdots$ END$>$においては、
$\mathbf{V}_{t_n}$END = true
$\mathbf{V}_{t_i}$END = false $(0 \leq i \leq n-1)$

通常の時間論理では、□、◇、until を基本としているが、これらのオペレータによる表現では、区間という概念が陽には現れず、自然言語の意味処理といった我々の目標とするような分野の問題の表現では、until の入れ子が幾重にもなり非常に理解しずらい。

3．3．　記述例

この節では、この論理を用いた例を与える。until を用いた方法では表現が複雑になる問題でも、簡単に表現することができる事が可能である。

（1）学生時代は、夏休みにはいつも山登りに行った人がいる。

$\exists x$[学生(x) $\rightarrow\sim$学生(x)]
[夏休み(x) $\rightarrow\sim$夏休み(x)]◇山(x)

（2）太郎は、初めて海にいった夏休みの次の夏休みは、ずっと海にいた。

$<<$夏休み（太郎）$\rightarrow\sim$夏休み（太郎）$>$
◇海（太郎）$\overset{①}{\rightarrow}$ false$>$
$<$夏休み（太郎）$\overset{①}{\rightarrow}\sim$海（太郎）$>\Box$海（太郎）

（3）花子は、初めて海に行った夏休みに読んだ本を、今も持っている。

$$\exists x(\ (<\text{START} \rightarrow <\text{夏休み}(\text{花子}) \rightarrow \sim\text{夏休み}(\text{花子}) > \Diamond \text{海}(\text{花子}) > <\sim\text{夏休み}(\text{花子}) \leftarrow \text{END} > \Diamond \text{読む}(\text{花子},x)\ ) \wedge \text{持つ}(\text{花子},x) \wedge \text{本}(x)\ )$$

（4）皆の給料は減ることはなかった。

$$\forall x\ \forall y(\ [\text{給料}(x,y) \rightarrow \text{給料}(x,z) \wedge x \approx z\ ]_{y<z})$$

3．4．　恒真式と略記法

この節では、様相オペレータを含む恒真な式の例を与え、省略記法を導入する。

3．4．1．　恒真式

以下の式は恒真である。

$\Diamond\Diamond G \equiv \Diamond G$

$\Box\Box G \equiv \Box G$

$\Box(A \supset B) \supset (\Box A \supset \Box B)$

$\Diamond<A \rightarrow B>G \equiv <A \rightarrow B>G$

$\Box[A \rightarrow B]G \equiv [A \rightarrow B]G$

$[A \rightarrow B]C \wedge [A \rightarrow B]D \equiv [A \rightarrow B](C \wedge D)$

$[A \rightarrow B](C \supset D) \supset ([A \rightarrow B]C \supset [A \rightarrow B]D)$

$([A \rightarrow B]D \wedge [B \rightarrow C]D) \supset [A \rightarrow C]D$

$[A \rightarrow \sim A]\Box A$

3．4．2．略記法

3．3の例題を見ればわかるように、イベント指向の考え方をすると、　　$P \rightarrow \sim P$

の形をした区間指定の形が頻繁に使用されることがわかる。これをわざわざ毎回書くのは非常に煩雑であるので、次のマクロ記法を導入する。

$$\text{while } P \text{ 並 } P \rightarrow \sim P$$

したがって、

$[P \rightarrow \sim P]$ を [while P] と、

$<P \rightarrow \sim P>$を<while P>と

書くことが可能である。

4．Templogにおける時間の概念

Templog では、現在システムが保持している節の集合に、assertあるいはretract によって、何らかの変更が加えられた際に、時間が進むと考える。

従って、Templog では、次のように節の履歴が保持されることによって時間が進む。

(1) 最初は、Templog のシステムには何もassertされていない。

(2) ある時点において、節の集合がassertされるので、Templog のシステムはこれを現在の状態S_0 とし、その時の時刻をTemplog システムの現在の時刻t_0とする。これが本当の意味での、初期状態である。

(3) システム時刻t_i で、現在保持している節の集合に変更が加えられると、Templog における時間が1つ進んだものとみなし、新しい状態S_{i+1} を作りだし、これを現在の状態とし、現在の時刻をシステム時刻t_{i+1} とする。

以後、Templog のシステムは(3) のプロセスを繰り返し、最も最近に作り出された状態が「現在」に対応する状態となり、バックトラックしても、時刻が戻ることはない。各々の時刻は、様相論理学で言うところの「可能世界」に対応するとも考えることができる。

従って、Templog システムにおける時間の概念は離散的であり、線形に順序付けられている。

このように節の履歴を管理するとすると、

(1) 関心のある事のみをassertすればよい、つまりイベントが起った、あるいはイベントが終了した、ということのみに注目すればよいから、絶対時間の概念は現れない。つまり、要求された時間の精度が保証されることなり、論証する時間の単位が任意であることが保証される、また

(2) Assertされた節は、retarct されるまでデータベース上に存在するから、持続性の概念がサポートされる。

4．時間論理プログラミング言語Templog

4．1．　TemplogのSyntax

Templog は、前章で定義された一階の区間を基にした時間論理に基づいているが、Prologと同様に、Horn節に準じた形のみを許している。

1) 述語

1-1) 基本述語は述語である。

1-2) $A_{i_1},\ldots,A_{i_{ni}}$を述語、あるいは基本述語とし$\alpha$をそれらの連言又は、それに～を付けたものとすると、（つまり、$\alpha_i = A_{i_1},\ldots,A_{i_{ni}}$ 又は、$\alpha_i = A_{i_1},\ldots,A_{i_{ni}}$（ただし、$n_i = 1$ のときは、$\sim A_{i_j}$ と書く））以下のものは述語である。

$<\alpha_1 \rightarrow \alpha_2 > \alpha_3$

$<\alpha_1 \textcircled{1}\rightarrow \alpha_2 > \alpha_3$

$[\alpha_1 \rightarrow \alpha_2\ ]\alpha_3$

$<\alpha_1 \leftarrow \alpha_2 > \alpha_3$

$<\alpha_1 \leftarrow \textcircled{1} \alpha_2 > \alpha_3$

$[\alpha_1 \leftarrow \alpha_2\ ]\alpha_3$

$\Box\alpha_1$

$\Diamond\alpha_1$

2) プログラム（節）

A_0を述語、$A_1,\ldots,A_n$ を述語あるいは！（カットシンボル）とすると、以下はプログラムである。

A_0 <= .　　(a)

A_0 <= $A_1,\ldots,A_n$.　　(b)

A_0 <- .　　(c)

A_0 <- $A_1,\ldots,A_n$.　　(d)

<- $A_1,\ldots,A_n$.　　(e)

Templog で扱う節には、

(1) 時刻とは独立なもの、
つまり、いつでも成り立つ節、および

(2) 履歴として考えるべきもの

の２種類に分類することが可能である。

(1) は、(a)、(b) の形のプログラムで表現し、例えば、次のappendのプログラムが挙げられる。

i) append([],X,X) <= .
append([X | Xrest], Y, [X | Zrest]) <=
append(Xrest, Y, Zrest) .

これはどの世界でも普遍的に成立すると考えるのが自然である。

(2) は、(c)、(d) の形のプログラムに対応する。

i) rich(john) <- .

これは、Johnという特定の人物が裕福であるということを示している述語であり、特定の世界でしか言えない事である。

ii) rich(X) <- sal-of(X,Y), Y ≧ 10k .

これは、ある人X がrichであるとはどういう事かを述べている述語であるが、これは社会情勢の変化等によって定義が変わることもありうる。

4．2．Templogの宣言的意味

自由変数は全称的に全体で束縛されると考え、(a)、(b) のプログラムの意味はそれぞれ□A_0．□ ($A_0 \subset A_1 \wedge \ldots \wedge A_n$) なる論理式の意味に対応し、(c),(d) のプログラムの意味はA_0，$A_0 \subset A_1 \wedge \ldots \wedge A_n$なる論理式の意味に対応する。

4．3．Templogの手続き的解釈

Templog の手続き的セマンティクスは、Prologのそれと基本的な部分においては同一である。すなわち、探索の戦略はtop down、depth first であり、またユニフィケーションやバックトラックの機構も、基本的には同一である。

プログラム中の変数に一度インスタンシエートされた値は、異なる時刻における評価でも保持される。但し、無名変数_は別であり、時刻ごとに異なる値がインスタンシエートされてよい。したがって、□P (X) の反駁が失敗する場合でも、□P (_) の反駁が成功することがありうる。

(実行制御)

(1) 反駁しようとする述語Pが様相オペレータを持っていない場合

i) まず、現在の状態で成立する非様相述語を頭部として持つ節の頭部と整合させる。これはPrologの実行の枠の内である。

ii) 整合できる述語が存在しなければ、

□G

の形の述語を頭部として持つ節が存在し、PとGとが整合可能ならば、Pと□Gは整合したものとする。

iii) さもなければ、次に、

[A → B] □G

の形の述語を頭部として持つ節を探し、PとGとを整合させる。成功すれば、この時得られた変数の代入の下で、

<<～A ←① END> ◇A ←① END>□～B

を評価し、これが成功すれば整合したものとみなす。つまり現在の時刻がA → Bで指し示される区間の中に含まれているかどうかを調べることと等価である。

iv) 以下同様の、様相記号がネストしたものを頭部として持つ節を探す。例えば、

[X → Y] [A → B] □G

を探し、PとGとが整合できれば、現在の区間がA → Bに含まれ、かつX → Yに含まれていればPは成功したとする。

(2) 次に、反駁しようとする述語に様相オペレータが含まれている場合についてのプログラムの実行制御について述べる。

まず、反駁しようとする述語と同じ形の述語を頭部に持つ節を探し、頭部同志を整合させる。ここでの整合は、様相記号を含めた頭部全体を1つのパターンとしてみなして行なう。成功すればこの時の変数の束縛の下でこの節の右辺の反駁に移る。このような節が存在しない、あるいは反駁に失敗すれば、以下のように実際に過去の履歴を参照する。

i) □Gの実行

評価中の区間の最初の時刻を始点とする区間を将来に向けて順にとり、Gを評価する。途中で失敗すれば、1つ区間を戻し、Gに対するオルタナティブについて評価を行なう。最初の区間におけるすべてのバックトラックについて失敗すれば、□Gの評価は失敗する。すべての区間でGが成功すれば、□Gは成功する。バックトラックが起り、□Gの評価に制御が戻った場合は、最後の区間でのGに対するオルタナティブについて評価を続ける。

例えば、次のような履歴が記録されているものとしよう。

t_1	t_2	t_3
rich(james) .	rich(john) .	rich(bill) .
rich(john) .	rich(mary) .	rich(john) .

この状態で、

<- □ rich(X).

という、「ずっと裕福だったのは誰か。」を尋ねる質問を発すると、X=johnのみで成功する。

ii) ◇Gの実行

□Pを頭部に持つ節があり、GとPとの整合が成功し、右辺も成功すれば、◇Gは成功する。

次に、i) と同様の部分区間で順次Gを評価する。途中で成功すれば、◇Gは成功とする。バックトラックが起り、◇Gの評価に制御がもどった場合には、先に成功した区間についてGのオルタナティブの評価を続ける。あらゆる区間についてGが失敗すれば、◇Gの評価は失敗する。

例えば、先の例の下で、

<- ◇ rich(X) .

という質問を発すると、

X = james　　X = john

X = john　　X = mary

X = bill　　X = john

の計６回成功する。

iii)　＜A → B＞Gの実行

まず、［C → D］Pを頭部に持つ節を探し、AとC、BとD、PとGとが同時に整合可能であれば、その右辺を評価し成功すれば全体が成功したものとする。

次に、i）と同様の部分区間で順次Aを評価する。Aがどの部分区間においても失敗するならば、全体も失敗する。Aが成功したなら、その区間I_1を全体の評価区間Iから引いた残りの区間I_2（但し、I_1の最後の時刻は、I_2の最初の時刻である）の将来へ向けての部分区間について順次Bを評価する。Bの評価が初めて成功した場合の区間I_3について、又はBの評価がすべての部分区間について失敗したならI_2について、Gを評価し成功すれば全体は成功する。

iv)　＜A ①→ B＞Gの実行

以下の点を除いて、＜A → B＞Gの実行とほぼ同様である。すなわち、Aに関するすべてのバックトラックに失敗した時、Aを評価していた次の部分区間について同様の評価を行なってゆくが、以前の部分区間でAが成功した時の変数の代入と同じ代入について成功しても、これは失敗とみなす。

v)　［A → B］Gの実行

＜A → B＞Gの実行とほぼ同様であるが、ある区間でGが成功した後も、この時のGのみに出現する変数の代入を保持したまま、Aの評価の区間を１つ進め、同様の処理を行ない、全ての区間I_3についてGが成功するならば、全体が成功する。あらゆるバックトラックを行ない、ある部分区間I_3についてGが成功する場合が無ければ全体は失敗する。

４．５．　Ｎｅｇａｔｉｏｎ　ａｓ　Ｆａｉｌｕｒｅ

Templog の～は、Prologの～と同様に閉世界仮説に基づいている。したがって、Prologにおいて、述語Pの真偽値と述語～～Pの真偽値とが必ずしも一致する訳ではないように、Templog においても、Pと～～Pとは同等ではない。

次に、◇Pと～□～Pとの等価性、および□Pと～◇～Pとの等価性について考察してみよう。ここでは、簡単の為に、先程の例での<-～□～rich(X).の実行を考えてみよう。ここで、Xがインスタンシエートされている場合と、されていない場合とに分けて考える事とする。

(1) Xがインスタンシエートされている場合。

まず、Xがjames にインスタンシエートされている場合を考えよう。

◇rich(james) が t_1 で成功するのは明らかである。～□～rich(james) の実行を考えてみると、まず～rich(james) が失敗し、□～rich(james) が失敗する。よって全体の節が成功する。

次に、Xがjackにインスタンシエートされている場合を考えてみよう。。

◇rich(jack)が失敗するのは明らかである。～□～rich(jack)の実行を考えてみると、□～rich(jack)の実行が成功するので、全体の節の実行は失敗する。

これからわかるように、一般的に反駁しようとする述語に含まれる変数がすべてインスタンシエートされている場合は両者は等価である。

(2) Xがインスタンシエートされていない場合。

まず、◇rich(X) を実行した場合には、前述したように全部で6 回成功する。

しかし、～□～rich(X) を実行した場合には、様子が異なる。まず、～rich(X) が t_1 で評価されるが、失敗する。よって□～rich(X) も失敗し、最終的に全体が成功するが、X に値がインスタンシエートされない。

同様に、□Pと～◇～Pとも、Pに変数が含まれる場合には、等価ではない。

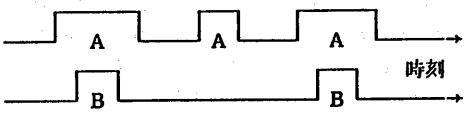
５．区間の抽象化

'２．１で述べたように、我々の時間に関する知識は特有の構造を持っており、その構造に従った、時間に関する知識のより高度な抽象化が必要であると考えられる。Templog には、区間に関する抽象化を行なえるような機構が含まれている。まず，区間に関してsuper-intervalとsub-intervalの概念を与える。

5.1 Super(Sub) interval

定義：区間Bが絶えず区間Aの間(during)にあれば、AはBのsuper-intervalであると言い、逆にBはAのsub-intervalであると言う。（図－１）

この場合区間の始まりと、終りは一致する必要はない。



図－１　Super(Sub)-interval

ここで、区間の間にsuper/sub-interval関係があるかどうかを示すメタ述語sub-int を導入する。

記法：区間Aが

P → Q

で表わされ、区間Bが

R → S

で表わされるものとする。

メタ述語sub-int は次のように記述される。

sub-int(P → Q , R → S) <=A_1,..., A_n .

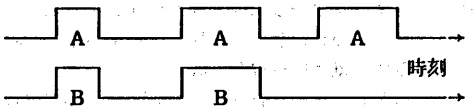
これは、A_1,...,A_n が成立すれば、P → Qで表わされる区間AがR → Sで表わされる区間Bのsuper-intervalである（つまり、BがAのsub-intervalである）ことを言っている。

さて、次にsub-interval class, super-interval class関係の概念を与える。

5.2 Super(Sub) interval class

定義：区間Bが常に区間Aの一つとみなせるとき、AはBのsuper-interval classと言い、逆にBはAのsub-interval classと言う。（図－2）

この場合、区間Bの始点、終点は、区間Aのそれと一致する。



図－２．Super(Sub)-interval class

記法：このような関係は次のメタ述語により表現される

sub-int class(P → Q , R → S) <=A_1,..., A_n

これは、A_1,....,A_nが成立すれば、P → Qで表される区間は、R → Sで表わされ区間とみなせることを宣言する。

5.3 評価実行

このような区間の間の二つの関係に関して、次の事が言える。

(1) [P → Q] □Gならば [R → S] □G

(2) <R → S>◇Gならば<P → Q>◇G

前者は例えば、「学生時代にずっと貧乏であれば、大学生時代もずっと貧乏である。」といった類の推論に対応し、後者は例えば、「大学生時代に海外に出たことがあれば、学生時代に海外に出たことがあることになる。」といった類の推論に対応する。

このような節は、評価において次のようにして用いられる。[R'→ S'] □ Gの評価に失敗した場合、 RとR'およびSとS'がそれぞれ整合するようなsub-int 定義又は，sub-int class 定義が存在すれば、その整合において得られた変数代入をもって、sub-int 定義の右辺 A_1,...,A_n を評価し、成功すれば、この時得られた変数代入の下で、[P →Q] □ Gを評価する。

同様にして、< P' →Q'>◇G の評価に失敗した場合、同様の変数代入のもとで、<R →S >◇G を評価する。

また、特にsub-int class 定義の下では、

[R'→ S'] G,< P' →Q'>G の処理においても同様の処理がおこなえる。以下に、これらの例を挙げる。

5.4 例

(1) 節 [while student(X)] □poor(X) <=. がassertされているものとする。ここで、

<- [while univ(john)] □poor(john) .

という質問が発せられた場合には、まず

[while univ(john)] □poor(john)

[while student(X)] □poor(X)

とを整合させようとするが、失敗し、システムは自動的にsub-int 関係を探し、

sub-int(while student(X),
while univ(X)) <= .

がassertされていれば、

システムはここで改めて述語

[while student(john)] □poor(john)

を反駁し、成功する。

(2) 学生時代はテストを受けることがあるということをを言っている節

<while student(X)>◇test(X) <= .

がassertされているものとする。

ここで、

<- <while univ(X)>◇test(X) .

なる質問を発した場合に、

<while student(X)>◇test(X) と

<while univ(X)> ◇test(X)

との整合には失敗するが、(1) と同様のsub-int 述語が検索され、最終的に成功する。

この例では、単一の述語で表現される、単純なイベント同志の関係であるから、student(X) <= univ(X) なる節を用意しておけば同じ事が行なえる。しかし(1) の例や、もっと複雑なイベント同志の関係では区間のsuper/sub-interval関係を表わすことが必要である事が多い。

(3) 次のような例を考えよう。

```
sub-int class (start-move(X,Vehicle)→
                    end-move(X,Distance),
      on-board(X,Origin,Vehicle)→
        off-board(X,Destination,Vehicle)) <=
    Distance = distance(Origin,Destination) .
```

ここで、述語の意味は以下の通りであるとする。

start-move(X,Vechicle)
: X がVehicle によって移動を開始した。

end-move(X,Distance)
: X がDistanceの移動を終了した

on-board(X,Origin,Vehicle)
: X がOriginで、Vehicle に乗った。

off-board(X,Destination,Vehicle)
: X がDestination で、Vehicle から降りた。

このようなsub-int 関係を定めた上で、

```
on-board(john,tokyo ,train) <-
off-board(john,fukuoka,train) <-
```

がこの順でassertされているとき、

[start-move(X,Vehicle)→end-move(X,Distance)]
tired(X) <= Distance>1000, Vehicle=train

の下で、

<-◇tired(john) .

を実行すると、成功する。

６．まとめ

区間概念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Temploq を提案した。本プログラミング言語は履歴に関する知識を取りあつかう事が可能である。すなわち、過去の経験の時間的関係を保持し、それを用いた推論が可能である。単純に時刻を対象とするよりも、区間を考える方が、我々の時間に対する直感によく整合し、時間概念に関する事実をより簡単で整理された形で記述することができる。Temprog では、区間の入れ子関係を表わすsub-intaval,および区間集合のclass ／sub-class 関係を表わすsub-interval classという二つの概念を導入した。

自然言語文の意味処理に時間概念を無視することが出来ないことが明らかになってきているが、本システムの開発は、自然言語文に対する意味処理に使用することを主たる目的としており、今後、さらに応用例を考える事により、基礎となる論理及び言語仕様の拡張が必要となろうし、区間論理の証明に関する研究も必要である。

謝辞：日頃、御指導いただく槇本肇教授に深く感謝いたします。

参考文献：

[1] J.A. Bubenko, Jr.: "On concepts and strategies for requirements and information analysis", SYSLAB Report No.4, Department of Computer Science, Charlmers University of Technology (1981)

[2] P. Needham: "Temporal intervals and Temporal order", Logic and Philosophy (1981)

[3] D. McDermott: "A Temporal logic for reasoning about processes and plans", Cognitive Science 6 (1982)

[4]Hayes, P.J.: "In defence of logic",In Proc. 5th International Joint Conference on Aritificial Intelligence

[5] J.F. Allen: "Maintaining Knowledge about Temporal Intervals", CACM Vol.26 Num.11 (Nov. 1983)

[6] N. Yonezaki and H. Enomoto: "Database System based on Intensional Logic", Proc. of 8th COLING (1980)

[7] 山口純一、米崎直樹、槇本肇：履歴データベースに対する関数型質問言語」、情報処理学会第25回全国大会、pp 585-586 (1982)

[8] J.Clifford, and D.S. Warren: "Formal semantics for time in database", ACM TODS Vol 8. No.2 (Jun. 1983)

[9] 米崎直樹、新淳、蓬莱尚幸：「時間論理プログラミング言語Temploq 」，日本ソフトウェア科学会第一回大会論文集、pp77～80 (1984)