プロセスを値として持つ論理 2ITL

2ITL: Logic which has process as a value of a variable

河野 真治
Shinji Kono E-Mail: kono@ie.u-ryukyu.ac.jp

琉球大学 情報工学科
Information Engineering, Univesity of the Ryukyus

概 要

2 階区間時相論理 (2ITL) について考察する。2ITL を用いた仕様記述により、従来、時相論理では表現できなかった、プロセス代数的な再帰や、メカニズムを持ったタイムアウトなどの記述ができる。2ITL は決定可能ではないが、モデルに対する制限を入れることによって構文的な制限を入れることなく決定可能にすることができる。その自動検証、そして、2ITL の実行法を示す。

1 2 階時相論理とは?

2ITL とは 2 階区間時相論理 (2nd order Interval Temporal Logic) のことである。本論文では真 (T) と偽 (F) のみを値とする 2ITL の命題論理の自動検証と実行を議論する。2 階論理 (2nd order logic) とは、論理式そのものを値とする変数を含む論理である。

2ITL はプロセス代数 [4] に相当する記述力を持ち、通常の命題時相論理と異なり、2 階の変数により、メカニズムを持つ fairness や eventuality、記述することが出来る。残念ながら本論文で示すように 2ITL は決定手続きは無い。しかし、モデルに対する制限を行なうことにより、そのシンタックスを制限すること無く、決定手続きを得ることが出来る。これにより、2ITL を直接に実行することが可能になる。

通常の古典命題論理の範囲内では 2 階の拡張は自明である。何故なら、自由変数の値が決まってしまえば、命題式の値は T/F しかなく、2 階の変数は式のどこに表れても同じ値なので、2 階の変数は命題変数とまったく同等である。この状況は、for all, exists などの量化記号があっても変わらない。

しかし、時相論理の場合は、同じように式の値は T/F しかないが、様相演算子の中にある式の値は時間 (この場合は時区間) に依存する。従って、2 階の命題変数そのものが様相演算子のような働きをする。つまり ITL そのものを 2 階の論理と考えることができる。本論文は full propositional ITL の制限された決定手続きを求めている。後で述べる Local ITL は、2 階の論理とみなすことはできない。

1 階論理 (1st order logic) または、述語論理は述語/関数を含む論理であり、1 階 ITL 区間時相論理の実行と検証に関しては、既にさまざまなシステムの実現ととその応用がなされている。1 階述語論理はプログラミング言語としてすぐれていて、普通の (例えば C の) プログラムを楽に記述できる。

しかし、同期関係にのみ注目するには記述力が高すぎて自動検証は難しい。その難しさの一つの理由は、1 階述語の範囲内で同期関係そのものが記述できるので、同期関係を時相論理の範囲で閉じさせることができないからど考えられる。

2 時区間時相論理 ITL

ITL は二つの様相演算子しか持たない論理である。

chop $P\&\ Q$ 時区間を二つに分け、前半で P が成り立ち、後半で Q が成り立つ。

next $@P$ 次の時刻から始まる時区間で Pが成り立つ。時区間が終わってしまって、 次の時刻がない 時には F。

ここで、ITL の時間は、0 から始まる離散的な時間で、無限に続くが時間の終わりはあるとする。つまり、compact descrite time を仮定する。これにより、無限に続く時区間を特別扱いする必要がなくなる。

さらに、これらを組み合わせて以下のような様相演算子を定義することもできる。これらにより、ほとんどのリアルタイムプログラミングを、プログラム言語のように記述することができる。

empty 時区間の長さが $0 \equiv \neg@T$

length(n) 時区間の長さが n。

somtime $<>P$ その時区間の中で、いつか P。$\equiv T\&\ P$

always $[]P$ その時区間の中で、いつも P。$\equiv \neg(T\&\ \neg P)$

all-always $\boxed{\text{a}}\ P$ その時区間の中の、どの時区間上でも P。$\equiv \neg(T\&\ \neg PT)$

ITL の命題変数 (P,Q,R... と大文字で表す) は T, Fの値を持つが時区間によって値が変わる。これを 2 階の変数と考えられることを後の章で示す。一方、時区間の終わりに依存しない性質を local という。

local(P) $P \equiv (P\&\ T)$

local な変数 (p,q,r... と小文字で表す) は、時刻にのみ依存するイベントを表す。通常の時相論理の変数はこのようなものであることが多い。すべての変数を local 変数とした ITL を LITL (local ITL) と言う。LITL の自動検証に付いては [3] で述べた。LITL のモデルは有限状態機械 (FSA) であり、任意の LITL 式に対応する FSA を作ることができる。

3 時相論理とプロセス代数を結ぶ 2 階の変数

もし、2 階の変数 P,Q,R がなんらかのプログラムを表すとすれば、それは FSA(=正規表現 Regular Expression, 以下 R.E.) を表していると考えられる。また、これらの変数を ITL の演算子で結合しても、それは FSA を表している。つまり、2ITL は R.E. について閉じている。従って、変数の表す時区間が正規表現に限られるという時区間論理を考えることができる。これを RITL(Regular ITL) という。LITL のモデルが FSA であることを考えれば、RITL を 2ITL と考えることができる。

プロセス代数では、プロセスを表す記号を定義する。2ITL は離散時間を持つプロセス代数を翻訳することができる。例えば、

$$\boxed{\text{a}}\ (P \rightarrow ((a \wedge\ \ ) \vee (b \wedge \neg a \wedge \text{empty}))$$

は、$P \equiv a.P|b$ に相当する。このように 2ITL は再帰を表現することもできる。これは今までの命題時相論理では不可能だったことである。もちろん 2TIL では Axiom Schema を直接表現することができる。

4 2 階の変数による仕様記述

2 階の変数はメカニズムと考えることもできる。

$$<>_S P \equiv S\&P$$

$<>_S P$ は、Sというメカニズムを持つ eventuality である。このメカニズムは以下のように実際に定義することができる。

$$\boxed{\text{a}}\ (S = length(2))$$

メカニズムとして任意の FSA を 2ITL 式を使って定義することができる。ただし、そのためには、量化記号が必要である。LITL は R.E. よりも弱いことを証明することができる。

5 ITL の表現力と決定性

もともと ITL は、プロセス代数のように通常のプログラムに近い表現を可能にするように作られている [5]。しかし、ITL(区間時相論理) は、もともと命題論理の範囲内でも決定手続きがないことが知られている [2]。これは、以下の式を使って簡単に証明することができる。

$$\boxed{\text{a}}\ ((P\&@Q) \rightarrow (R\&@S))$$

この ITL 式は、$PQ \rightarrow RS$ という文脈依存文法 (CFG) を表している。文脈依存文法の充足問題は決定不能なので、ITL の充足問題も決定不能である。

ところが RITL も決定手続きを持たない。何故なら、ある CFG が Regular かどうかを示す問題も決定不能であることが知られているからである [1]。従って、2ITL の決定手続きは、正規表現よりも強い制限を付けなくては存在しない。以下に、その制限に付いて考察する。

6　モデルの制限と決定性

finite length　2ITL の制限のもっとも強い制限は、2 階の変数の T/F の変化に時間制限を付けることである。これにより、2 階の変数の状態数を有限にすることができる。時区間自身は無限長で良いが、ある長さ m を越えるとその値は変わらない。これは以下の式によって表すことができる。

$$\text{limit(P)}\ \ (length(m)\&true)\ \rightarrow\ (P\ \equiv\ ((P \wedge length(m))\&\ T))$$

finite process　次に弱い制限は、2 階の変数の表すプロセスの数に制限を付けることである。式の表すプロセスの数は時間によって変化する。このプロセスの数が m を越えると、もっとも古いプロセスの値は T/F に固定されるとする。これは、操作論的な制限になる。

finite interraction　最後に、もっとも弱い制限として、2 階の変数の相互作用の数に制限を付けることができる。異なる時区間上の 2 階の変数は異なる値を持つ。ある時区間上に単独に現れる 2 階の変数は相互作用しないので、非決定的に T/F になる様相演算子に置き換えることができる。逆に言えば、2 階の変数は自分自身としか相互作用しない。相互作用を持つプロセスの数が m を越えると、もっとも古いプロセスの値は T/F に固定されるとする。これも、操作論的な制限である。

これらの操作論的な制限は、Tableau expansion rule として定義することができる。これらの制限に抵触せずに、2ITL 式の unsatisfiability を示せれば、その否定は、これらの制限がなくても valid であることは簡単にわかる。(制限に抵触しない場合の 2ITL Completeness) また、m を徐々に増加させることにより、2ITL の部分決定手続きを得たことになる。RITL 式が充足可能ならば、m を増加させることにより必ず解を見つけることができる。

LITL 自身が (量化記号を仮定して)R.E. を含むので、これらの制限により、2ITL が R.E. よりも弱くなることはない。また、2ITL が CFG を含むこともない。この決定手続きは、トークンの長さを制限した CFG などなどの決定手続きに相当する。

7　2ITL の tableau expansion

2 解の変数 R の tableau expansion は以下のように表すことができる。

$$\begin{aligned} R &= (\text{empty} \wedge R_0) \vee @R_0 \\ R_n &= (\text{empty} \wedge R_n) \vee @R_{n+1} \end{aligned}$$

この$_n$が、どんどん大きくなることが 2ITL/ITL の undecidability を表している。 finite length は、この$_n$を制限することに相当する。

例えば、

state 1　$R \wedge @@@@@@@@@@\text{empty}$

という 2ITL 式は limit 5 では、以下のように展開される。

state 2:　$R_1 \wedge @@@@@@@@@\text{empty}$
state 3:　$R_2 \wedge @@@@@@@@\text{empty}$
state 4:　$R_3 \wedge @@@@@@@\text{empty}$
state 5:　$R_4 \wedge @@@@@@\text{empty}$
state 6:　$R_5 \wedge @@@@@\text{empty}$
state 7:　@@@@empty
state 8:　@@@empty
state 9:　@@empty
state 10:　@empty
state 11:　empty

finite process は、tableau expansion 中に出てくる R_nの$_n$を順序を変えずに番号を 0 からふりなおすことに相当する。これにより、$T\&\ R$ などの右再帰をモデルを制限に抵触することなく構築することができる。

finite process は、tableau expansion 中に出てくる R_n が、その時区間上で唯一であるもの (singleton) を非決定性を表す様相演算子に置き換えること (と再番号付け) によって実現できる。これは一種の 3 値論理になる。singleton かどうかを決めることは自明ではないが、それを決める手続きが存在することを示すことができる。この制限により、$P\&\ T$ などの左再帰式も制限抜きにモデルを作ることができるようになる。

最後の finite interraction は、2ITL 式に対するモデルの制限としては実用的であると考えられる。もちろん、決定手続き自身は、指数計算量を要求するので、計算量的に実用的ということはできないが、十分広い範囲の論理式を制限に到達することなくモデルを構築できるということである。

8 2 階時相論理の実行

2ITL の決定手続きは、モデルとして FSA を与える。従って、その FSA の実行が 2ITL の実行ということになる。実際には、2 階の変数は、その終了時刻を R_n という local な変数によって表現される。前の例では、

```
| ?- ex((^r,(length(10) = ^r))).
0.46299999999999963 sec.
11 states
16 subterms
23 state transitions
counter example:
0:+r^0    false
| ?- exe.
execution:
0:      2
1:      3
2:      4
3:      5
4:      6
5:+over(r,5)    7
6:      8
7:      9
8:      10
9:      11
10:     0
yes
```

というように実行される。`over(r,5)` が、2 階の変数 R が制限を越えて値 T に固定されたことを示している。

その他の制限を採用した場合には、実行した場合の local な変数などは正しく再現されるが、2 階の変数の実行は、再番号付けなどにより、直接には見ることができない。しかし、実行時に制限をしない tableau expansion をおこない、得られたモデルと対応する状態を選択することにより、2 階の変数の実行を再現することができる。これを詳細実行という。

これは、未だに決定されてない部分の存在するプログラムを実行するという今までにない方法を示している。また、再帰を含む同期機構を命題命題論理の範囲内で表現し、詳細実行したという点が新しい。

9 将来の課題

もちろん、ここで示した自動検証法は計算量的には実用的ということはできない。しかし、FSA は得られるので小さい仕様を使ったモデル検査には直接しようすることができると思われる。

Finite interraction における詳細実行は tableau expansion の再計算を必要とする。これを不要にすることはできないのだろうか?

この論理では、2 階の変数は時刻ごとに異なる値を持つ。時刻で変わらない値を持つような 2 階の変数を持つ論理の自動検証は可能なのだろうか?

2ITL は、ITL のモデルを制限したものであり、構文的な制限は存在しない。ここであげたモデルの制限の性質を調べることは重要だろうと考えられる。いったい、どのような 2 階論理式が制限に抵触せずに証明できるのか、必要な制限 n を予想することはできるのだろうか? また、さらに緩い制限を見つけることができるのだろうか?

モデルの制限は実際のリアルタイムプログラムに対するプログラミング方法論的な指針にもなっている。これらの指針を実際のプログラムに生かすことができるような枠組みを作ることが重要だと考えられる。

参考文献

[1] J. E. Hopcroft and J. D. Ullman. *Introduction to automata theory, language and computation.* Addison-Wesley, 1979.

[2] Shinji Kimura and Shuzou Yajima. Formal languages satifing temporal logic formula. *IECE Japan*, Vol. J70-D, No. 1, pp. 117–123, 1987. in Japanese.

[3] Shinji Kono. A Combination of Clausal and Non Clausal Temporal Logic Program. In *Executable Modal and Temporal Logics*, volume LNAI-897. Springer-Verlag, 1994. Lecture Notes in Artifificial Intelligence.

[4] R. Milner. *A Calculus of Communicating Systems*, volume 92 of *Lecture Note in Computer Science*. Springer-Verlag, 1980.

[5] B.C. Moszkowski. Executing temporal logic programs. Technical Report 55, Computer Laboratory, Univ. of Cambridge, 1984.