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

Shinji Kono
Information Engineering, Univesity of the Ryukyus
E-Mail: kono@ie.u-ryukyu.ac.jp

Abstract: A process is a value of a variable of higher order interval propositional temporal logic (2ITL here after). Not only temporal relation-ship among events, but also processes are directly defined in terms of temporal logic. The process includes finite state machine, fairness, a scheduling mechanism, inverse specification, and various temporal logic formula.

1 Second Order Temporal Logic

In normal temporal logic, variables represents events depending on a clock period like other Temporal Logics. But in Interval temporal logic (Ref. [3] ITL here after), variables represent series of events in intervals of time. Value of 2nd order variable is an arbitrary formula. We call ITL with 2nd order variables 2ITL.

In case of classical logic, second order propositional logic is trivial, because the value of the 2nd order variable is either T or F. For a formula $f(P)$ with 2nd order variable P and propositional variable p, we can prove

$$\vdash f(p) \text{ iff } \vdash f(P).$$

There are no differences between second order logic and normal logic from the view point of validity.

The same situation happens in ITL.

$$\vdash f(P) \text{ iff } \vdash f(p) \text{in ITL.}$$

If-part is trivial becase p is a possible interpretation of P. In the other direction, for an interpretation $M_{\sigma_1...\sigma_n}$ of Kripke structure, truth value of P is fixed, $M_{\sigma_1...\sigma_n}(P)$. $\sigma_1...\sigma_n$ is n length interval. This is a truth value fuction of $\sigma_1...\sigma_n$. If we use this fuction for defining truth value of p, $f(p)$ is true from the assumption. So $f(P)$ is true.

This proof works only on full ITL, which is undecidable[3]. A possible restriction of ITL is local ITL. In local ITL, truth value of every variable is determined at the first clock of the interval. Clearly we cannot have the truth value fuction of $\sigma_1...\sigma_n$ for a local variable. This means 2nd order variable have to be an interval variable, which value changes on each interval.

Since full ITL is undecidable, in this paper, we'll show some restrictions which makes ITL decidable.

2 Specification Examples

A second order variable represents a mechanism to terminate an interval. It is possible to think it a fairness.

$$\Diamond_S P \equiv S \& P$$

$\Diamond_S P$ means P is eventually true on fairness S, eventuality-S. Unlike fairness in LTTL, many kinds of different fairness can be defined in 2ITL. In fact the value of second order variable is different on each clock period, so it defines different fairness on each clock period. We can think this is an abstraction of watch dog timer or counter. The mechanism of the timer can be defined in terms of ITL;

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

$\boxed{\text{a}}\ P$ means P is true on all sub-interval. This means S assures an interval which is less than 2 clocks. $less(n)$ operator can be defined using n-times nested weak next operator; $less(2) \equiv \bigcirc\bigcirc$empty. This is a simple watch dog timer mechanism (Fig. 1).

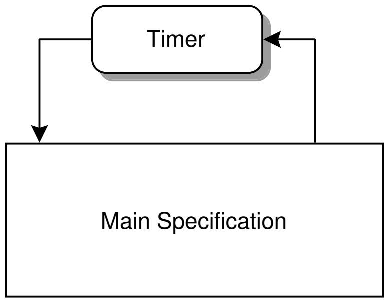


Figure 1: External Timer

Of course, we can use second order variables to define a process by recursion like process algebra.

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

But we have to consider the restrictions of second order variables in these process representations in case of verification.

3 Undecidability from a View Point of Tableau Expansion

In this section, we discuss on a tableau method of 2ITL. In Tableau method on discrete temporal logic, temporal logic terms are decomposed into two parts. One part is depending only on current clock period and the other part only depends on next or later interval.

Here we show an example of decomposition of a chop operator. Assume P, Q is already decomposed into empty parts PE, QE, current clock dependent parts PN_i, QN_i and next interval dependent parts PX_i, QX_i in disjunctive normal form.

$$\begin{array}{lcl} P & = & PE \wedge \text{empty} \vee \bigvee_{i=0}^{k}(PN_i \wedge @PX_i) \\ Q & = & QE \wedge \text{empty} \vee \bigvee_{i=0}^{k}(QN_i \wedge @QX_i) \end{array}$$

Where @ is next operator with ¬empty and it is called strong next. Then $P\&Q$ is decomposed in this way.

$$\begin{array}{lcl} P\&Q & = & (PE \wedge Q) \vee \\ & & \bigvee_{i=0}^{k}(PN_i \wedge @(PX_i\&Q)) \end{array}$$

In practical tableau expansion, BDD standard form and deterministic expansion is necessary, but these are discussed in (Ref. [1, 2]).

In case of second order variable or interval variable, the value of the variable is depend on the both begin-time and end-time of the interval. Since we are working on propositional case, it looks like we can replace the value of the variable by T or F. Yes, the value is T or F, but the value depends on the interval (Fig. 2). So we cannot replace the second order variable with T or F as we did in classical logic.

This situation is demonstrated by an example:

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

If we replace R by T or F, this example becomes unsatisfiable. But this is satisfiable if

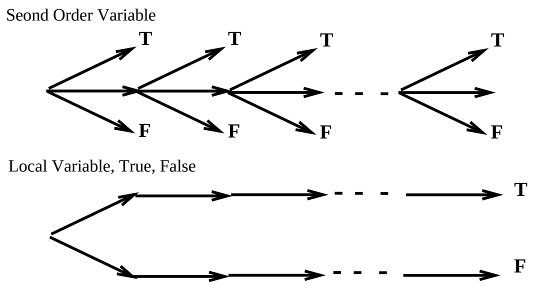
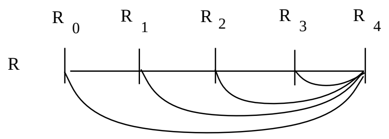


Figure 2: Value of second order variable

we instantiated R by $length(2)$ or equivalent FSM.

If we think R as a FSM, tableau expansion should have next form.

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



R_n is n-th state of R. R_n is also a second order variables. R_n is independent each other if n is different. This is because R_n has different start point and R has different truth value in different interval. Using existential quantifier on second order variable, we can write;

$$R \quad = \quad (\text{empty} \wedge R) \vee @\ \exists S\ S.$$

During the tableau expansions, n increases infinitely. Actually n represents the interval length of R. In case of a formula like $\Box R$, infinitely many R_n are generated. In this way, undecidability of full ITL or 2ITL happens in the tableau expansion.

3.1 Length Restriction and Count Restriction

The simplest stopper of the undecidability is length limit.

$$\begin{array}{lcl} R & = & \text{empty} \wedge R_0 \wedge @R_0 \\ R_n & = & \text{empty} \wedge R_n \wedge @R_{n+1} \text{ if } n < k \\ R_n & = & beg(R_n) \text{if } n \geq k \end{array}$$

R behaves normal in less than length n interval and after the limit it is fixed to T or F.

R_n is generated on every clock. But if we have only one R_n, we don't have to use specific n. Any other number is also ok. We can compact the number sequence by sort and renaming. But the order of the number have to be preserved. After renaming, n represents number of R_n in a formula. We can call this restriction count limit. The count limit is useful on a formula like this:

$$R \& T.$$

In this example, R_n increases n by 1. This generates a series like this.

$$R \& T, R_1 \& T, R_2 \& T, ... R_n \& T$$

If we think R and R_1 are equivalent, this example is expanded to itself. Roughly speaking, in count limit method, we have no limit on one time R-eventuality.

3.2 More Complex Restrictions

But above restrictions does not work well on a formula like $T \& R$ that is $\Diamond R$. In this example, expanded formula has a form after n clocks;

$$R_0 \vee R_1 \vee ... \vee R_n \vee T \& R.$$

After R_n reaches the limit, it becomes T or F. In this case, renaming of R_n becomes identity and useless. Looking at the formula carefully, we find every R_n exists only once. The meaning of this formula is not depends on the particular name of R_n and R_n is independent each other. It is possible to remove R_n from the formula. This is called singleton removal.

Here we show several restriction methods on decidable 2ITL;

length limit Effects of R has time limit,

count limit Number of R is limited,

singleton removal Number of interrelated R is limited.

These are defined in an operational way in the tableau expansion.

Computational complexity of 2ITL verification is determined by the restriction. Local ITL verification requires exponential complexity of the length of the formula, that mostly comes from determination of expanded states. In the worst case, all combination of the sub terms have to be computed. R_n terms increase the number the sub term, as result in effect of 2^n. In our experience, count limit is slightly faster than length limit.

4 Execution of 2nd Order Interval Temporal Logic

In the tableau expansion based verification (Ref.[2]), a deterministic FSM is generated. This is a method of logic synthesis or program generation. In case of Local ITL, all variables are events. An execution of 2ITL is simple. Besides conditions on events variable, it also contains conditions on second order variables. The conditions depend on the restriction methods.

Length limit is the most simple one. It contains two kind of events on R.

termination R is terminated in T or F in less than length n interval.

time out the limit of R expired and results T or F.

These are marks on the FSM and define FSMs for R on each clock period. It also define a trace of R in the execution.

In case of count limit, we have to consider renaming of R_n. Other situation is similar to the length limit case. The renaming is assigned to each transition in the generated FSM. Using renaming information, we can find a FSM definition of R on each clock period.

4.1 Example: Length Operator and Second Order Variable

We can demonstrate the difference of length limit and count limit by using simple example: $R \wedge (R = length(10))$. Here we assume limit is 5. $length(10)$ is expressed by nested strong next operator and this becomes the first state.

state 1 $R \wedge$ @@@@@@@@@@empty

In case of length limit, it is expanded in this way;

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

In state 5, R_6's truth value is fixed. R_6 is false entire formula is false otherwise @@@@empty remains. The resulted state diagram is show in (Fig.3).

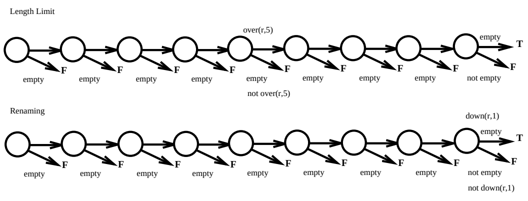


Figure 3: FSM for Length Example

4.2 Example: Grammar Rules or Recursive Process

A 2ITL formula,

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

represents a grammar rule or CCS like process. But we are using discrete time and it use $\boxed{a}$ operator, the verifier generates rather big FSM.

```
| ?- ex(
(^r,'[a]'(( ^r -> ((a,@ ^r);
   (~a,b,empty)))))).
76.817 sec.
128 states
22 subterms
264 state transitions
yes
| ?- exe(5).
0:+a-r^0 3
1:+a-r^0-r^1 67
2:+a-r^0-r^1-r^2 99
3:+a-r^0-r^1-r^2-r^3 115
4:-a+b+r^0+r^1+r^2+r^3+r^4 0

yes
| ?- exe(10).

no
```

`+a` means `a` is T and `-a` means `a` is F. `exe(10)` requests at least length 10 example and it has no solution because of count limit 5. The solution should be a Regular Expression `a*b`, but we cannot find out the solution because of the undecidability of 2ITL.

5 Comparisons

Several methods has been developed for real-time specification.

- Timed automaton
- Process Algebra
- Higher Order Logic
- Temporal Logic

Timed automaton can be a basic model of real-time specification. Process algebra provides a program oriented syntax by recursion style. Temporal logic provides natural language like and declarative syntax of specification.

In process algebra, all processes are defined by recursions. Propositional temporal logic cannot describe recursions, but 2ITL can. Chop operator's role is very important here. If we use LTTL and Until operator, even if we use second order variable, recursive process is not expressed. But recursion is not a perfect because of our second order variable restriction.

Corresponding part of verification in process algebra is complex hierarchy of bi-simulation. Since 2ITL is logic, failure set and success set is compliment of each other. This corrupts the hierarchy of bi-simulation, and it is defined as temporal logic relation.

6 Future direction

Current method works well on small examples, but we cannot say it is practical. It requires huge computation to verify second order variable and the expressiveness of the variable is restricted. This is because we are concentrated on a FSM based automatic verification and easy restriction. One possible direction is to construct theorem prover on for example HOL. The other direction is to find out more practical restrictions.

References

[1] Masahiro Fujita and Shinji Kono. Synthesis of Contrllers from Interval Temporal Loigc Specification. *International Workshop on Logic Synthesis*, May 23-26, 1993.

[2] 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 Artifficial Intelligence.

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