Temporal Logics

Angelo Montanari (and Alberto Policriti)
Dipartimento di Matematica e Informatica,
Università di Udine

Plan

1) classification

2) linear time temporal logics: $PLTL$

3) branching time temporal logics: CTL and CTL^*

4) model checking

Modal Logic	Temporal Logic
$\Box$ / $\Diamond$	(always/sometimes)
$\texttt{TRUE} \rightsquigarrow$ *necessarily* $\texttt{TRUE}$	$\texttt{TRUE} \rightsquigarrow$ *always* $\texttt{TRUE}$

Classification: parameters

propositional vs. first-order; global vs. compositional (formalism)

branching vs. linear; points vs. intervals; discrete vs. continuous (time)

past-future vs. future only

...

most fashionable (useful, reasonable, ...)

propositional/global/point-based/future-tense

Linear Time

Propositional Linear Temporal Logic (PLTL)

$PLTL$: syntax

$P, Q, ...$ propositional letters (AP)

$\wedge, \neg, ...$ propositional connectives

X(next), U(until) temporal connectives

Formulae: $P, p \wedge q, \neg p, Xp, p\ Uq$

Shorthands:

$F\ p \equiv true\ U\ p$ (eventually p)

$G\ p \equiv \neg F \neg p$ (always p)

$F^{\infty} p \equiv GF\ p$ (infinitely often)

$p\ B\ q \equiv \neg((\neg p)\ U\ q)$ (p *before* q)

$\underline{PLTL\text{: structures}}$

$M : (S, x, L)$ discrete time/ initial instant / infinite in the future (cf. M. O. Rabin, *"Decidable Theories"*, in Handbook of Mathematical Logic, 1977)

S set of *states*

$x : \mathbb{N} \to S$ *sequence* of states

$L : S \to Pow(AP)$ *labeling* function

$x \equiv (s_0, s_1, \ldots) \equiv (x(0), x(1), \ldots)$ *fullpath, computation sequence, computation, ...* (may seem useless!).

Notation: for all $i = 0, 1, \ldots$, let $x^i = (s_i, s_{i+1}, \ldots)$ (in particular, $x^0 = x$).

$\underline{PLTL}$: semantics

M : (S, x, L) *linear time structure*

Definition of truth:

$$M, x \models p$$

(i.e. p is true in M at $x(0)$: modal in nature)

$$\begin{array}{rcl}
M, x^i \models P & \Leftrightarrow_{\text{def}} & P \in L(s_i) \\
M, x^i \models p \wedge q & \Leftrightarrow_{\text{def}} & M, x^i \models p \text{ and } M, x^i \models q \\
M, x^i \models \neg p & \Leftrightarrow_{\text{def}} & M, x^i \not\models p \\
M, x^i \models p \; U \; q & \Leftrightarrow_{\text{def}} & \exists j \geq i(x^j \models q \wedge \\
 & & \wedge \forall i \leq k < j(x^k \models p)) \\
M, x^i \models X \; p & \Leftrightarrow_{\text{def}} & M, x^{i+1} \models p
\end{array}$$

Remarks

1. $x \models p$ is as to say $x(0) \models p$ (more generally, $x^i \models p$ is as to say $x(i) \models p$);

2. (Important) the semantics of the modal operators is a first-order formula in a language whose individual variables range over states;

3. (In our formulation) we adopted a *strong*, *non-strict* version of the until operator, denoted $p\ U_{\exists}^{\geq}\ q$.

   Many variants of it have been defined.

   Most important: *strict* (*strong*) until (denoted $p\ U_{\exists}^{>}\ q$). $X\ q$ ($= X(false\ U_{\exists}^{\geq}\ q)$) can be defined as $false\ U_{\exists}^{>}\ q$.

Variants of Until - 1

<u>weak until</u>: p holds for as long as q does not, even forever if need be.

It is defined as:

$$x \models p \; U_{\forall} \; q \quad \Leftrightarrow_{\text{def}} \quad \forall j (\forall k \leq j (x^k \models \neg q \rightarrow \\ \rightarrow x^j \models p))$$

or, in terms of $p \; U_{\exists} \; q$ and of the derived operator $G \; p$, as:

$$x \models p \; U_{\forall} \; q \quad \Leftrightarrow_{\text{def}} \quad x \models p \; U_{\exists} \; q \vee G \; p$$

<u>strong until</u>: there does exist a future state where q holds and p holds until then

$$x \models p \; U_{\exists} \; q \quad \Leftrightarrow_{\text{def}} \quad x \models p \; U_{\forall} \; q \wedge F \; q,$$

Variants of Until - 2

where $F\ q \Leftrightarrow_{\mathsf{def}} \neg(\neg q\ U_{\forall}\ false)$

(and thus $G\ q \Leftrightarrow_{\mathsf{def}} (q\ U_{\forall}\ false)$).

<u>Remark</u>: weak and strong until operators are *inter-definable*.

<u>strong strict until</u>: strong + future $\not\subseteq$ present

$$x \models p\ U_{\exists}^{>}\ q \quad \Leftrightarrow_{\mathsf{def}} \quad \begin{array}{l} \exists j > 0(x^j \models q \wedge \\ \wedge \forall k < j(x^k \models p)) \end{array}$$

Kamp theorem: *The Monadic First-Order theory of discrete linear orders with first element is equivalent to $PLTL$ (with strong strict until).*

cf. H. Kamp *"Tense logic and the theory of linear orders"* PhD thesis, UCLA, 1968.

What about the *Past*?

$$X^- \; ; \; p \; U^- \; q \; (S \text{ } \textit{Since}) \; ; \; F^- \; (P) \; ; \; G^- \; (H)$$

Adding past operators allows one to extend Kamp theorem to discrete linear orders (and beyond)

cf. D. Gabbay, A. Pnueli, S. Shelah, and J. Stavi *"On the temporal analysis of fairness"* 7th ACM Symposium on Principles of Programming Languages, 1980.

Branching Time

A state may have *many* successor states (i.e. consider many linear time models *at once*).

Structures will become *trees*

CTL (Computational Tree Logic): structures

$M : (S, R, L)$ *branching time structure*

S set of *states*

$R \subseteq S \times S$ *binary relation* on states

$L : S \rightarrow Pow(AP)$ *labeling* function

with such a general R, M is a graph rather than a tree: unfold it!

The problem is the language !

CTL: language

X and U can be seen as *quantifiers* over states in a computation (cf. the semantics)

Expressive power is enhanced adding *quantifiers over computations*:

$$\begin{array}{rcl} A & : & \text{for all futures} \\ E & : & \text{there exists a future} \end{array}$$

$$\begin{array}{rcl} X, U, ... & : & \textit{state} \text{ quantifiers} \\ A, E & : & \textit{path} \text{ quantifiers} \end{array}$$

CTL forces (syntactically) two path quantifiers to be interleaved with one state quantifier.

$\underline{CTL^\star}$ no restrictions (unifying framework for branching and linear temporal logics)

The syntax of CTL is given by defining *state formulae* and *path formulae* (depending on the most external operator):

(s_1)	$P \in AP$	state f.lae
(s_2)	p, q state f.lae $\Rightarrow$	$p \wedge q$ $\neg p$ state f.lae
(s_3)	p path f.la $\Rightarrow$	$A\ p$, $E\ p$ state f.lae

(p_0)	p, q state f.lae $\Rightarrow$	$X\ p$, $p\ U\ q$ path f.lae

$CTL^\star$ is obtained by replacing (p_0) by (p_1),(p_2), and (p_3):

(p_1)	state f.lae $\Rightarrow$	path f.lae
(p_2)	p, q path f.lae $\Rightarrow$	$p \wedge q$ $\neg p$ path f.lae
(p_3)	p, q path f.lae $\Rightarrow$	$X\ p$, $p\ U\ q$ path f.lae

CTL ($CTL^\star$) formulas
$=$
the set of *state* formulas.

Variants of $CTL^\star$:

- $PCTL^\star$: extension of $CTL^\star$ with past operators (over paths);
- $DCTL^\star$: extension of $CTL^\star$ with (explicit) successors;
- $PDCTL^\star$: extension of $CTL^\star$ with both past operators and successors.

Semantics: two notions of truth

1) $M, s_0 \models p$ for p state formula
2) $M, x \models p$ for p path formula

state-formulae semantics:

$$\begin{array}{rll}
M, s_0 \models P & \Leftrightarrow_{\text{def}} & P \in L(s_0) \\
M, s_0 \models p \wedge q & \Leftrightarrow_{\text{def}} & M, s_0 \models p \text{ and } M, s_0 \models q \\
M, s_0 \models \neg p & \Leftrightarrow_{\text{def}} & M, s_0 \not\models p \\
M, s_0 \models E\ p & \Leftrightarrow_{\text{def}} & \exists x = (s_0, ...)(M, x \models p) \\
M, s_0 \models A\ p & \Leftrightarrow_{\text{def}} & \forall x = (s_0, ...)(M, x \models p)
\end{array}$$

path-formulae semantics:

$$\begin{array}{rll}
M, x \models p & \Leftrightarrow_{\text{def}} & x = (s_0, ...), p \text{ is a state-f.la,} \\
 & & \text{and } M, s_0 \models p \\
M, x \models p \wedge q & \Leftrightarrow_{\text{def}} & M, x \models p \text{ and } M, x \models q \\
M, x \models \neg p & \Leftrightarrow_{\text{def}} & M, x \not\models p \\
M, x \models p\ U\ q & \Leftrightarrow_{\text{def}} & \exists i (M, x(i) \models q \wedge \\
 & & \wedge \forall j < i (M, x(j) \models p)) \\
M, x \models X\ p & \Leftrightarrow_{\text{def}} & M, x(1) \models p
\end{array}$$

Basic Issues

Given the syntax and semantics of a temporal logic (either linear or branching), one faces the following issues:

- what properties can be expressed: expressivity
- existence of calculi: axiomatizability
- decidability and complexity issues:
  - satisfiability/validity
  - **model checking**

Model checking is (by far) the most popular among the studied problems.

It is simple, computationally "affordable" , central to verification, and ... standard for industrial applications.

E. Clarke, E. Emerson, and A. Sistla *"Automatic Verification of finite state concurrent systems using temporal logic"*, Proc. of the 10th ACM Symp. on Principles of Programming Languages, 1983.

General idea: instead of considering the full satisfiability problem (given a formula φ, is there a model for φ?) consider

checking the truth of φ in a given M

Model checking can be *much* less complex than satisfiability (think of SAT).

It was originally presented for CTL with the following (simplified) syntax:

$$P, \neg p, p \wedge q, AXp, EXp, A(p\ U\ q), E(p\ U\ q)$$

It is outlined for input structures (S, R, L) with S finite. If S is infinite, some kind of **abstraction** is necessary.

Remark (cf. also the tableaux technique): in order to understand if p is true at a given state, we need to know if *any* subformula of p is true at any other state (there is room for improvements/optimizations)

General idea (for the model checking algorithm):

- associate a set of *labels* with each state (i.e., the set of sub-formulae true at that state);

- initialize the set of labels in a given state with atomic propositions (looking at the input);

- proceed inductively on the structural complexity of formulae to extend the set of labels: 7 recursive calls to a procedure `label-graph`

$$P, \neg p, p \wedge q, AX\ p, EX\ p, A(p\ U\ q), E(p\ U\ q),$$

where the last two are the non-trivial cases.

- $A(p\ U\ q)$: from a given state, start with a search for a state in which q holds along *each* possible fullpath (and guarantee that p holds until then).

Use a stack to implement a depth-first search.

Mark states to avoid cycles.

- $E(p\,U\,q)$: (simpler) start from states in which q holds and walk backward along paths thru which p holds

`label-graph` must be called on each subformula ... complexity on a formula p:

$$O(|p|(|S| + |R|))$$

linear in the size of the model.

Model checking is linear but:

1. it works for finite-state models only;

2. it does not implement any *fairness* condition;

3. it works for propositional logic only.

Model checking for $PLTL$ is more complicated (on branching models).

(Linear) Temporal Logic and ω-Languages

- Models of $PLTL$ are ω-strings α in a suitable alphabet (for each state, a character encodes the truth value of the propositional symbols on that state)

- the theory of formal languages can be extended to ω-languages: **Büchi automata**

- (non-deterministic!) finite state automata

- acceptance condition: $\mathcal{A}$ accepts α if and only if there is a *run* of $\mathcal{A}$ on α that passes infinitely often thru some final state

On this ground we define:

$$\mathcal{L}(\mathcal{A}) = \{\alpha : \mathcal{A} \text{ accepts } \alpha\}$$

$$\mathcal{L}(\varphi) = \{\alpha : \alpha \models \varphi\}$$

where φ is a $PLTL$-formula ($MFO[\leq]$-formula)

What is the relative expressive power of $PLTL$-formulae with respect to Büchi-automata acceptance?

$\mathcal{L}$ is the set of models of a $PLTL$ formula

if and only if

$\mathcal{L}$ is the set of models of an $MFO[\leq]$ formula

if and only if

$\mathcal{L}$ is accepted by a counter-free finite state automaton

First equivalence: Kamp theorem

Second equivalence: McNaugthon and Papert theorem

$\mathcal{L}$ is the set of models of an ETL formula

if and only if

$\mathcal{L}$ is the set of models of a $QPLTL$ formula

if and only if

$\mathcal{L}$ is the set of models of an $MSO[\leq](S1S)$ formula

if and only if

$\mathcal{L}$ is accepted by a finite state automaton

First and second equivalences: P. Wolper *"Temporal Logic can be more expressive"*, Information and Control, 1983 (satisfiability is elementarily decidable in ETL and non-elementarily decidable in $QPLTL$).

Third equivalence: Büchi theorem

Müller automata are the deterministic version of Büchi automata

Müller automata differs from Büchi automata in

- the set of final states:
  F set of final states $\rightsquigarrow$ a family $\mathcal{F} = \{F_i\}_i$

- the acceptance condition:
the set of states visited infinitely often belongs to $\mathcal{F}$

> *A finite automaton (either deterministic or non-deterministic) is the compact (more compact if non-deterministic) description of a family of computations of a finite state system.*

On the Expressiveness of $CTL^\star$

It has been shown that, when interpreted over infinite binary trees, $CTL^\star$, as well as $PCTL^\star$, is as expressive as $MSO[<]$ (where $<$ is the prefix order) with set quantification restricted to infinite paths.

By incorporating successors in both the computational tree logics and the monadic second-order 'path' logics, such a result can be generalized to $DCTL^\star$, as well as to $PDCTL^\star$.

Bibliography

A. Pnueli, *"The temporal logic of programs"*, IEEE Symp. on Found. of C.S., 1977.

T. Hafer and W. Thomas, *"Computation tree logic $CTL^\star$ and path quantifiers in the monadic theory of the binary tree"*, Proc. of the 14th International Colloquium on Automata, Languages and Programming (ICALP), LNCS Volume 267, Springer 1987.

A. Emerson, *"Temporal and Modal Logics"*, Chapter 16 of the Handbook of Theoretical Computer Science, 1990.

W. Thomas, *"Automata on Infinite Objects"*, Chapter 4 of the Handbook of Theoretical Computer Science, 1990.

Z. Manna and A. Pnueli, *"Temporal Verification of Reactive Systems: Safety"*, Springer, 1995.