Temporal Logic

[Language, Intuition and Possibilities]

Michael Fisher
Department of Computer Science, University of Liverpool, UK

[MFisher@liverpool.ac.uk]
In classical logic, formulae are evaluated within a single fixed world. For example, a proposition such as “it is Monday” is either *true* or *false*.

Propositions are then combined using constructs such as ‘∧’, ‘¬’, etc.

In temporal logics, evaluation takes place within a set of worlds. Thus, “it is Monday” may be satisfied in some worlds, but not in others.

The *accessibility relation* between worlds then describes a particular model of *time*.

The set of classical operators is extended with various *temporal* operators which navigate by this relation.
Commonly, propositional, discrete, linear *temporal* logic extends the descriptive power of propositional logic in order to be able to

- describe sequences (hence: *linear*)
- of distinct (hence: *discrete*) worlds,
- with each world being similar to a classical (propositional) model.

So, we can equivalently describe the basis of our model of time in terms of a sequence of worlds, a sequence of states, or a sequence of propositional models.

Each state in the sequence is taken as modelling a different moment in time; hence the name *temporal logic*. 
We use a simple temporal logic (PTL) where the accessibility relation characterises a discrete, linear order isomorphic to the Natural Numbers, \( \mathbb{N} \).

The typical temporal operators used are

<table>
<thead>
<tr>
<th>Operator</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \diamond \varphi )</td>
<td>( \varphi ) is true in the <em>next</em> moment in time</td>
</tr>
<tr>
<td>( \square \varphi )</td>
<td>( \varphi ) is true in <em>all</em> future moments</td>
</tr>
<tr>
<td>( \Diamond \varphi )</td>
<td>( \varphi ) is true in <em>some</em> future moment</td>
</tr>
<tr>
<td>( \varphi \cup \psi )</td>
<td>( \varphi ) is true up until <em>some</em> future moment when ( \psi ) is true</td>
</tr>
<tr>
<td>start</td>
<td>only true at the <em>beginning of time</em></td>
</tr>
</tbody>
</table>
Towards Temporal Logic
PTL “in a nutshell”
Past
Branching
Varieties
Working with PTL
Safety
Liveness
Fairness
Utility
What Next?

monday ⇒ ◯ tuesday
start ⇒ ◊ finish
july ⇒ ◊ (december ∧ winter)
send(msg, rcvr) ⇒ ◊ receive(msg, rcvr)
□((¬passport ∨ ¬ticket) ⇒ ◯ ¬board_flight)
sunset ⇒ ◯ (night Udawn)
born ⇒ ◊ □ old
monday ⇒ sad Usaturday
Looking Back

Towards Temporal Logic
PTL “in a nutshell”
Past
Branching
Varieties
Working with PTL
Safety
Liveness
Fairness
Utility
What Next?

- Past
- PAST
- FUTURE
- now

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>● ϕ</td>
<td>ϕ was true in the <em>previous</em> moment in time</td>
</tr>
<tr>
<td>■ ϕ</td>
<td>ϕ was true in <em>all</em> past moments</td>
</tr>
<tr>
<td>♦ ϕ</td>
<td>ϕ was true in <em>some</em> past moment</td>
</tr>
<tr>
<td>ϕSψ</td>
<td>ϕ has been true <em>since</em> some past moment when ψ was true</td>
</tr>
</tbody>
</table>
Alternative is to use a *branching* model of time.
**CTL Syntax:** each temporal operator is now prefixed by one of the following *path operators*

- **A** – ‘on all future paths starting here’
- **E** – ‘on some future path starting here’

**Examples:**

- \( \mathbf{A} \square \text{safe} \)
- \( \mathbf{E} \bigcirc \text{active} \)
- \( \mathbf{A} \lozenge \text{terminate} \)

**Varieties:**

<table>
<thead>
<tr>
<th>Logic</th>
<th>Typical Formula</th>
<th>Problems</th>
</tr>
</thead>
<tbody>
<tr>
<td>CTL</td>
<td>( A \lozenge (E \lozenge p \land E \square q) )</td>
<td>lacks expressiveness</td>
</tr>
<tr>
<td>CTL*</td>
<td>( A \Box \lozenge EAp )</td>
<td>complex</td>
</tr>
</tbody>
</table>

There are many other varieties also.
Many different models of time are used, all *constraining* the accessibility relation \((R)\) in some way, for example

**linear** — each moment has at most one future moment

**branching** — a moment may have several future moments

**discrete** — if \(R(w_1, w_2)\) then there is *no* \(w_3\) such that \(R(w_1, w_3)\) and \(R(w_3, w_2)\)

**dense** — if \(R(w_1, w_2)\) then there is *always* a \(w_3\) such that \(R(w_1, w_3)\) and \(R(w_3, w_2)\)

**finite past** — there exists a \(w_1\) such that we can find *no* \(w_2\) such that \(R(w_2, w_1)\)
Varieties of Temporal Operators

The vast range of different models lead to a large range of operators that can be seen in temporal languages:

- **standard discrete linear future** — \( \Diamond, \Box, [ \Box ] \)
- **interval future** — \( U, W \)
- **past** — \( \bullet, \lozenge, \blacksquare, S, Z \)
- **fixpoints** — \( \mu, \nu \)
- **path quantifiers** — \( A, E \)
- **quantified propositional** — \( \exists p \ldots \Diamond p \)
- **full first-order** — \( \forall x \ldots \Diamond p(x) \)
- etc....
Deduction
Various proof methods have been developed, e.g.

— tableaux
— non-clausal resolution
— sequent systems
— translation methods (to FOL)
— translation methods (to MSOL)
— clausal resolution
— interactive theorem-provers
...
Model Checking
Very popular technique for checking whether a temporal formula is satisfied in a particular (finite) structure
- the finite structure is often derived from program or hardware descriptions.

Automata
Temporal logics have a close relationship to \( \omega \) automata, i.e. finite state automata over infinite objects
- consequently automata-theoretic methods are often employed.
Temporal Logic Gets Everywhere

specification and verification of (dynamic) programs
specification and verification of distributed and concurrent programs
representation of tense in natural language
characterising temporal database queries
verification of finite state models derived from ‘real’ systems (model checking)
agent theory
direct execution
real-time analysis
temporal data mining
exploring the limits of decidability

........
Safety:

“something bad will *not* happen”

Typical examples:

- $\square \neg (\text{reactor\_temp} > 1000)$
- $\square \neg (\text{one\_way} \land \Diamond \text{other\_way})$
- $\square \neg ((x = 0) \land \Diamond \Diamond \Diamond \Diamond (y = z/x))$

and so on.....

Usually: $\square \neg$. ....
Intuition: Liveness Properties

Liveness:

“something good \textit{will} happen”

Typical examples:

\begin{itemize}
  \item \textit{rich}
  \item \textit{terminate}
  \item \((x > 5)\)
\end{itemize}

and so on.....

Usually: \(\Diamond \text{ ....} \)
Intuition: Fairness Properties

Fairness (strong):

“if we attempt/request infinitely often, then we will be successful/allocated infinitely often”

Often only really useful when scheduling processes, responding to messages, etc. Typical example:

$$\forall p \in \text{processes}. \Box \Diamond \text{ready}(p) \Rightarrow \Box \Diamond \text{run}(p)$$

There are many forms of fairness, e.g:

- $$\Box \Diamond \text{attempt} \Rightarrow \Box \Diamond \text{succeed}$$
- $$\Box \Diamond \text{attempt} \Rightarrow \Diamond \text{succeed}$$
- $$\Box \text{attempt} \Rightarrow \Box \Diamond \text{succeed}$$
- $$\Box \text{attempt} \Rightarrow \Diamond \text{succeed}$$
We might describe the properties we require of a simple resource allocation system using temporal logic:

\[ \Diamond (\exists x. \text{allocate}(x)) \]  
\[ \land \quad \Box \neg (\text{allocate}(a) \land \text{allocate}(b)) \]  
\[ \land \quad \forall y. \Box \Diamond \text{request}(y) \Rightarrow \Box \Diamond \text{allocate}(y) \]  

[**LIVENESS**]  
[**SAFETY**]  
[**FAIRNESS**]

**Aside:** here, each process/agent requests a resource using ‘*request*’ with its name as argument.

System allocates to process/agent \( p \) by ‘*allocate*(\( p \))’.
We can usefully specify, using temporal formulae, systems with
dynamic,
concurrent,
distributed,
.....etc.....
aspects.

For example, from the resource allocator specification earlier, we might:

1. given a program intended to implement a resource allocator, we might prove that the program’s specification implies some temporal requirements.
But that isn’t all:

2. given a program that implements such a resource allocation system, we might \textit{model check} a finite representation of the program to see if the implementation actually satisfies the required specification;

3. we might be able to \textit{synthesize} a program directly from the above temporal logic requirement; or

4. we might be able to \textit{directly execute} the above temporal specification to provide an implementation.
We can now cover a range of topics concerning temporal logics, such as:

- syntax and semantics of temporal logic;
- simple temporal specifications;
- temporal reasoning, using a *resolution* procedure;
- formal verification, using *model checking*; and
- executable temporal specifications.

The first one is essential; the second is very useful; while the rest are independent of each other.