T-79.231 Parallel and Distributed Digital Systems

Temporal Logic

Marko Mäkelä

January 20, 2003

Temporal logic

Even simple looking systems can easily have thousands or millions of reachable states and enabled transitions. It does not make sense to graphically represent or manually explore each node and arc of so large graphs.

Safety properties, such as deadlock freedom or the unreachability of a forbidden state, can be formulated as conditions concerning one state at a time, which a computer can verify when adding nodes to the reachability graph.

Formulating and verifying liveness properties (“there is progress in the system”) call for new methods, such as *temporal logic* and *model checking*.

Properties of systems

- safety: “the system never reaches a bad state”; in each state holds P
  - deadlock freedom
  - mutual exclusion etc.
- liveness: “there is progress in the system”; X occurs infinitely often
- fairness; once X has occurred, Y will occur in n steps
  - sent messages are eventually received
  - each request is served
- self-stabilisation: “the system recovers from a failure in a finite number of steps”

Representing time

Most properties on the previous slide can be formulated by combining two operations:

- *finally* in the future
- *globally* in the future

It must be chosen whether the present belongs to the future. Time can be described in several ways:

- global time or local time for each party
- linear or branching
- discrete or continuous

The time may be associated with the occurrences of events or with the state of the system at each moment.

Representing time: fixing the choices

- We use a discrete global time that is tied to the occurrences of events.
- The present belongs to the future.
- We mostly observe the states as a function of time, not the events.

One thing is difficult to solve: should the time be linear or branching?

LTL (linear temporal logic): the behaviour is a collection of *infinite* transition sequences

CTL (computational tree logic): the behaviour is an *infinite* transition tree

Each logic can express properties that cannot be represented in the other logic. The union of **LTL** and **CTL**, **CTL*** is even more *expressive*: it can express some properties that are beyond the power of both **CTL** and **LTL**.

Linear temporal logic LTL

- The state propositions or formulae (Φ: $p \in \Phi$ if p) map system states to truth values.
- The formulae $Fma(\Phi) \supset \Phi$ include state propositions and
  - the false proposition $\perp \in Fma(\Phi)$
  - implication: if $a \in Fma(\Phi)$ and $b \in Fma(\Phi)$ then $a \rightarrow b \in Fma(\Phi)$, and
  - the connective “globally”: if $a \in Fma(\Phi)$ then $\Box a \in Fma(\Phi)$.

Other connectives can be expressed using these:

$$\begin{array}{rclcrcl}
\neg a & \Leftrightarrow & a \rightarrow \perp & \quad & \Diamond a & \Leftrightarrow & \neg\Box\neg a \\
a \vee b & \Leftrightarrow & (\neg a) \rightarrow b & & \top & \Leftrightarrow & \neg\perp \\
a \wedge b & \Leftrightarrow & \neg(a \rightarrow \neg b) & & & &
\end{array}$$

This is just one way of defining LTL and its basic connectives.

Linear temporal logic LTL: examples

- Messages are always delivered: $\Box((s = 0) \rightarrow \Diamond(s = 1))$
- The philosophers 1 and 2 cannot eat simultaneously: $\Box\neg(\text{eating}_1 \wedge \text{eating}_2)$
- p only holds in finitely many states, in other words $\neg p$ will permanently hold after a finite transition period: $\Diamond\Box\neg p$
- p holds in infinitely many states: $\Box\Diamond p$
- Safety property: $\Box p$

Interpreting linear temporal logic LTL (1/3)

- LTL formulae are interpreted with respect to infinite state sequences.
- A pair of a state set S and a reachability relation $R \subseteq S \times S$ forms a *frame* $\langle S, R \rangle$.
- Each state has only one successor: $R : S \rightarrow S$
- The reflective and transitive closure R^* of $R \subseteq S \times S$ is the smallest superset of R for which the following conditions hold:
  - $\bigcup_{s \in S} \langle s, s \rangle \subseteq R^*$ (reflectivity)
  - $\{\langle s, t \rangle, \langle t, u \rangle\} \subseteq R^* \Rightarrow \langle s, u \rangle \in R^*$ (transitivity)
- A Φ-*model* is a triple $M = \langle S, R, T \rangle$ where $T : \Phi \rightarrow 2^S$ maps state propositions to state sets: $T(p)$ is the set of states where p holds.

Interpreting linear temporal logic LTL (2/3)

- The following inductive definition states when a formula a holds in the state $s \in S$ of the model M:

$$\begin{aligned} M \models_s p &\Leftrightarrow s \in T(p) \\ M &\not\models_s \bot \\ M \models_s (a \rightarrow b) &\Leftrightarrow (M \models_s a) \rightarrow (M \models_s b) \\ M \models_s \Box a &\Leftrightarrow \forall t \in S, \langle s,t \rangle \in R^* : M \models_t a \end{aligned}$$

- The formula a holds in the model M (denoted $M \models a$) if and only if it holds in the first state of the sequence $s_0 \in S$: $M \models_{s_0} a$.

Interpreting linear temporal logic LTL (3/3)

- The models are the state sequences $v_0, v_1, \ldots$ of a reachability graph $G = \langle V, E, v_0 \rangle$, starting from the root vertex $v_0 \in V$, such that $\langle v_i, v_{i+1} \rangle \in E$.
- Let $\mathcal{M}(G)$ denote the set of all these sequences.
- The formula $a \in Fma(\Phi)$ holds in the reachability graph G if and only if $M \models_{v_0} a$ for all $M \in \mathcal{M}(G)$.
- Since a cyclic reachability graph may generate an infinite number of models, this is not not a good method of checking properties.
- Next, we shall describe the basics of a method [2] to check whether an LTL formula holds.

Examples of LTL formulae and corresponding sequences

The following table depicts three infinite state sequences, showing the propositions holding in each state. After state 6, the propositions remain constant.

Formulae that hold	0	1	2	3	4	5	6	$\dots$
$\Diamond p, \Diamond\neg p, \Diamond\Box p, \Box\Diamond p$		p		p		p	p	$\dots$
$\neg\Box\Diamond p, \Box\Diamond q, \Diamond(p \to \Box q)$			p,q	q	q	p,q	q	$\dots$
$\neg\Diamond\Box p, \Box\Diamond q, \Box(p \to \Diamond q)$		p	q		p		q	$\dots$

Short formulae can be checked in small models without using a computer, by interpreting one temporal connective at a time, starting a new “loop” whenever a $\Box$ or a $\Diamond$ is seen.

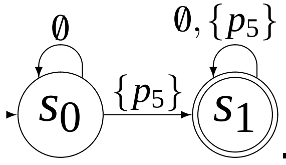
For instance, the formula $\Diamond(p \to \Box q)$ says that whenever p holds, also q must hold in that state and in all following states. The proposition p only holds in 2 and 5, and $\Box q$ holds in both states.

Model Checking (1/4)

- If the set of states is finite, the mapping $\mathbb{N} \to (\Phi \to \{\top, \bot\})$, or $\mathbb{N} \to 2^{\Phi}$ represents the model. This corresponds to an infinite sequence of characters $m \in 2^{\Phi}$.

- The LTL formula a defines a language on infinite sequences: the set of models where a holds.

- For instance, $\Diamond p_5$ holds in all sequences where p_5 sometimes holds. The alphabet of the language is $\Sigma = \{\emptyset, \{p_5\}\}$ and the words contain at least one p_5.

- The words of the language $\Diamond p_5$ can be detected with an automaton: 

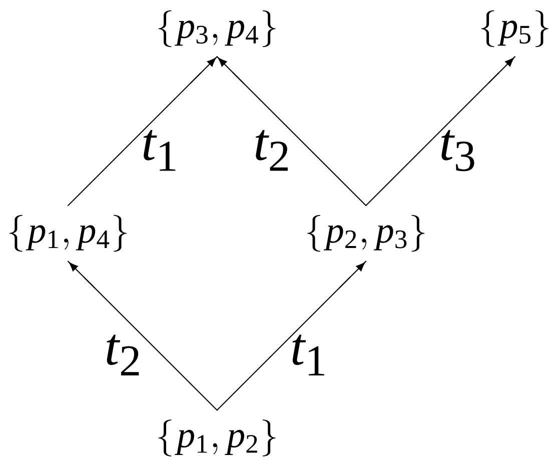
. The *initial state* is s_0, and s_1 is an *accepting state*.

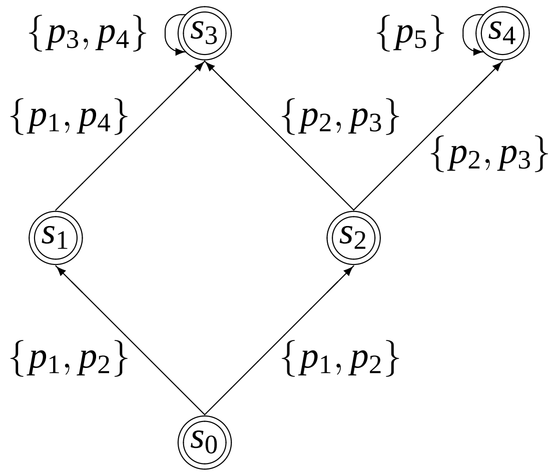
Model Checking (2/4)

- Because the words are infinite, the acceptance condition of finite automata does not apply (a word is accepted if it ends when the automaton is in an accepting state).
- The automaton Q accepts an infinite word $w = a_0, a_1, \ldots$ if and only if
  - there is an infinite sequence $\sigma = s_0, s_1, \ldots$ such that s_0 is an initial state of Q,
  - each s_{i+1} is derived from s_i and a character a_i via the transition function, and
  - there is an infinite number of integers $i \in \mathbb{N}$ such that s_i is an accepting state.
- The automaton Q is a *Büchi automaton*.
- Each LTL formula a can be translated into a Büchi automaton $B(a)$ that accepts exactly those infinite state sequences for which a holds.

Model Checking (3/4)

- A reachability graph G can be interpreted as a Büchi automaton $B(G)$ whose all states accept and whose transitions are labelled with the state propositions holding in the source vertex.

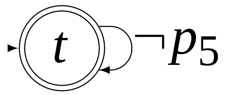




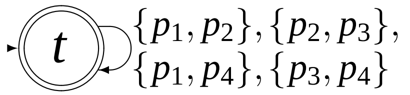
- The formula a can be model checked in G by computing the intersection or the product of the automata $B(\neg a)$ and $B(G)$. The formula holds if the product is empty.

Model Checking (4/4)

Let us model check $a = \Diamond p_5$ in the reachability graph of the previous slide. The Büchi automaton $B(\neg a) = B(\neg\Diamond p_5) = B(\Box\neg p_5)$ is as follows:



or adapted to the $B(G)$ on the previous slide

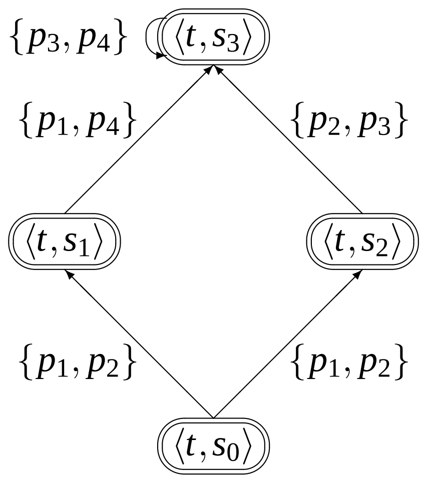


.

Let us compute the product $B(\neg a) \times B(G)$. How come, it is not empty! There are two *counterexamples* for the formula $\Diamond p_5$:

$$\{p_1,p_2\} \xrightarrow{t_1} \{p_2,p_3\} \xrightarrow{t_2} \{p_3,p_4\} \rightarrow \{p_3,p_4\} \rightarrow \cdots$$
$$\{p_1,p_2\} \xrightarrow{t_2} \{p_1,p_4\} \xrightarrow{t_1} \{p_3,p_4\} \rightarrow \{p_3,p_4\} \rightarrow \cdots$$

In the original model, the state $\{p_3, p_4\}$ is a deadlock, but the Büchi automaton is augmented with a loop, making all executions infinite.



Using the MARIA tool as a model checker

The MARIA tool model checks properties while it is constructing a reachability graph. If a property does not hold, MARIA will present a *counterexample*, that is, a transition sequence starting from the initial state, for which the property does not hold. This saves time compared to the approach of first constructing the whole reachability graph.

It does not make sense to use the heavy machinery of LTL for checking safety properties. The `reject` formulae of the MARIA language allows the specification of forbidden states as state propositions. Deadlocks can be detected with a `deadlock` formula.[*]

The LTL formulae to be verified are input in the MARIA query language and not as a part of the model, like the `reject` and `deadlock` formulae.

[*]Since LTL works on *infinite* executions, it does not know about deadlocks. Deadlock states are swept under the carpet by imagining self-loops into them.

LTL extensions (1/2)

Sometimes it is useful to speak of time intervals: “before”, “until”

$$\begin{aligned} M \models_s (a \,\mathsf{U}\, b) \Leftrightarrow \exists \langle s,t\rangle \in R^* : M \models_t b \wedge \\ \forall u \in S, \langle s,u\rangle \in R^*, \langle u,t\rangle \in R^* : (M \models_u a \vee u = t) \end{aligned}$$

The property that answers a must not precede questions q can be written $\Diamond a \longrightarrow (\neg a) \,\mathsf{U}\, q$.

The formula $(\neg q \,\mathsf{U}\, p) \vee \Box p$ states that a state where q holds must be preceded by a state where p holds.

The connective U is “stronger” than the connectives $\Diamond$ and $\Box$, because

$$\Diamond a \Leftrightarrow \top \,\mathsf{U}\, a \quad \text{and}$$
$$\Box a \Leftrightarrow \neg(\top \,\mathsf{U}\, \neg a).$$

Marko Mäkelä

LTL extensions (2/2)

The connective "in the next state" is often needed:

$$M \models_s \bigcirc a \Leftrightarrow M \models_t a \text{ where } \langle s,t\rangle \in R$$

For this connective, it holds:

$$\begin{aligned}\Box a &\Leftrightarrow a \wedge \bigcirc\Box a\\ \Diamond a &\Leftrightarrow a \vee \bigcirc\Diamond a\\ a \bigcup b &\Leftrightarrow b \vee \bigcirc(a \bigcup b)\end{aligned}$$

This leads to the following recurrence rules:

$$\begin{aligned}\Diamond\Box\neg a &\Leftrightarrow \Box\neg a \vee \bigcirc\Diamond\Box\neg a\\ &\Leftrightarrow (\neg a \wedge \bigcirc\Box\neg a) \vee \bigcirc\Diamond\Box\neg a\end{aligned}
\qquad
\begin{aligned}\Box\Diamond a &\Leftrightarrow \Diamond a \wedge \bigcirc\Box\Diamond a\\ &\Leftrightarrow (a \vee \bigcirc\Diamond a) \wedge \bigcirc\Box\Diamond a\end{aligned}$$

The expressiveness limits of LTL (1/2)

Let there be a property p that holds in each even state on all sequences starting from the initial state. It may assume any truth value in the odd states:

$$\begin{array}{ccccccccccccccc}
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \ldots \\
s_1 & = & p & \neg p & p & \neg p & p & \neg p & p & \neg p & p & \neg p & p & \ldots \\
s_2 & = & p & p & p & \neg p & p & \neg p & p & \neg p & p & \neg p & p & \ldots
\end{array}$$

For instance, the following sequences violate the requirement:

$$\begin{array}{ccccccccccccccc}
 & & 0 & 1 & 2 & 3 & 4 & 5 & 6 & 7 & 8 & 9 & 10 & \ldots \\
s_3 & = & \neg p & p & \neg p & p & \neg p & p & \neg p & p & \neg p & p & \neg p & \ldots \\
s_4 & = & p & \neg p & \neg p & \neg p & p & \neg p & \neg p & \neg p & p & \neg p & p & \ldots
\end{array}$$

The expressiveness limits of LTL (2/2)

For instance, the following formulae do not distinguish the traces in the desired way:

$$p \wedge \Box(p \rightarrow \bigcirc\neg p) \wedge \Box(\neg p \rightarrow \bigcirc p)$$
$$p \wedge \Box(p \rightarrow \bigcirc\bigcirc p)$$

Another, more general result holds:

A formula that contains state formulae and at most n $\bigcirc$ connectives has the same truth value for all sequences

$$\sigma_k = \langle \underbrace{p, p, \ldots, p}_{k}, \neg p, p, p, \ldots \rangle$$

where $k > n$.

References

1. Z. Manna and A. Pnueli: *The Temporal Logic of Reactive and Concurrent Systems: Specification*. Specifying systems with temporal logic; verifying them via proofs.

2. P. Wolper: *Temporal Logic*. Chapter 4 in A. Thayse, *From Modal Logic to Deductive Databases*. Model checking algorithms.

3. J. Esparza and S. Merz: *Model Checking* (lecture slides). A tutorial introduction to LTL, CTL and model checking algorithms.

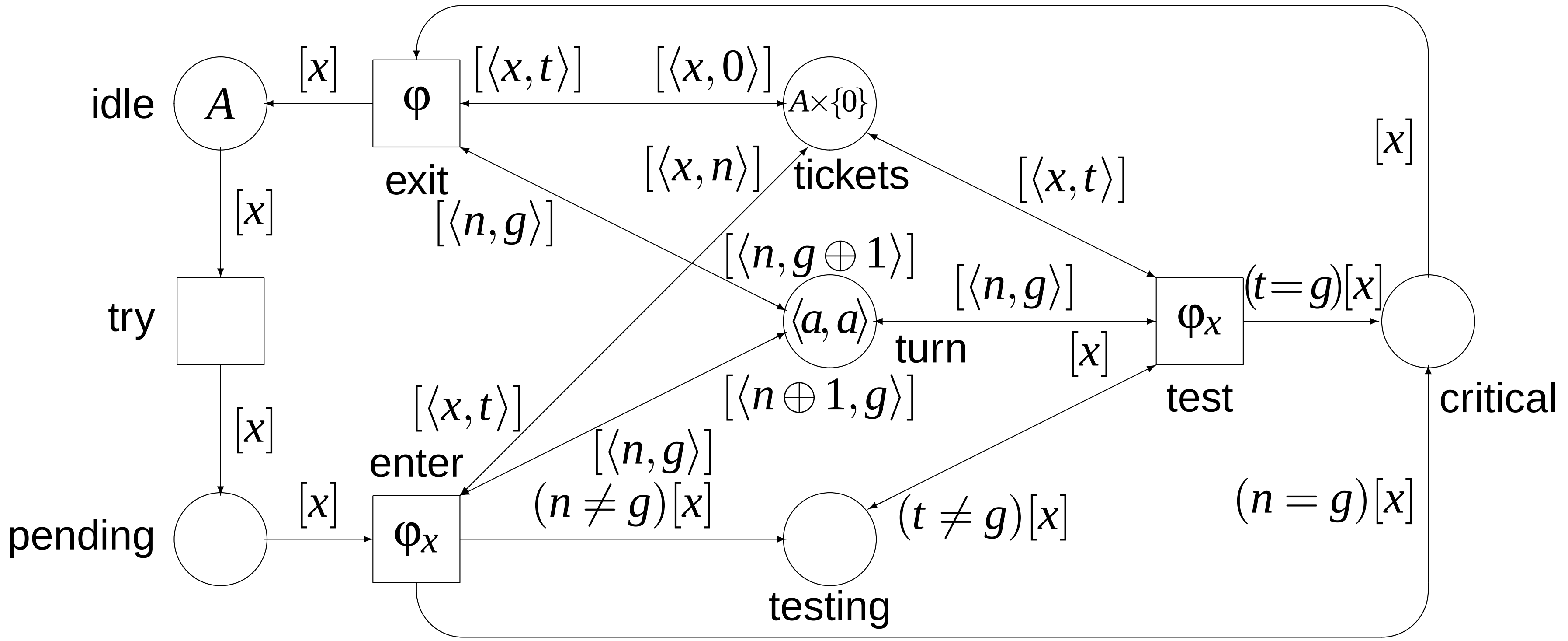
Example: mutual exclusion algorithm

Let us model check the following algorithm:

TicketME algorithm: A shared variable holds a pair $\langle \mathit{next}, \mathit{granted} \rangle$ of values in $\{1, \ldots, n\}$, initially $\langle 1, 1 \rangle$. The *next* component represents the next “ticket” into the critical section, while the *granted* component represents the last “ticket” that has been granted permission to enter the critical section. When a process enters the trying section, it “takes a ticket,” that is, it copies and increments the *next* component modulo n. When the ticket of a process is equal to the *granted* component, it goes to the critical region. When process exits the critical section, it increments the *granted* component modulo n.

Nancy A. Lynch: Distributed Algorithms, 1996, ISBN 1-55860-348-4

The algebraic system net of TicketME



Marko Mäkelä

Properties to check in TicketME

Let G be the reachability graph of the model for two processes ($A = \{1,2\}$). The following properties can be checked in it:

- At most one process is in the critical section: $G \models \Box\neg(\text{critical}(1) \wedge \text{critical}(2))$
- There is an execution where a process reaches the critical section (counterexample for the negation of the property): $G \not\models \Box\neg\text{critical}(1)$ or $G \not\models \Box\neg\text{critical}(2)$
- A process trying to enter the critical section will eventually get there (does not hold!): $G \models \Box(\text{pending}(1) \rightarrow \Diamond\text{critical}(1))$ or $G \models \Box(\text{pending}(2) \rightarrow \Diamond\text{critical}(2))$.

The last property fails to hold in an execution where the first process enters the critical section and the second one must wait. Now if the first process leaves the critical section and tries to get there again, it will not be able to proceed unless the execution of the second process resumes. Some *fairness assumptions* are needed.

Fairness (1/3)

If a system may execute infinitely many actions without executing an enabled transition, the reachability graph of the system will contain infinite sequences where the transition is constantly “neglected.” The concept of *fairness* has been defined for infinite sequences:

weak fairness: a constantly enabled event must occur infinitely often ($\Diamond\Box e \rightarrow \Box\Diamond o$)

strong fairness: an event that becomes enabled infinitely often (and may become disabled) must occur infinitely often ($\Box\Diamond e \rightarrow \Box\Diamond o$)

A fairness assumption may cover one transition or a set of transitions. In the latter case, the requirement is that if the set contains constantly (or infinitely often) enabled transitions, one transition belonging to the set must be fired infinitely often.

If there are several fairness assumptions, the counterexamples must treat each fairness set fairly as described above.

Fairness (2/3)

The MARIA tool allows fairness sets to be defined both for different instances of a single transition and for different instances of different transitions. In order for the formulae $G \models \Box(\text{pending}(x) \rightarrow \Diamond\text{critical}(x))$ to hold for all $1 \leq x \leq n = |A|$ processes, $n+1$ weak fairness assumptions are needed.

The set φ: Weak fairness must be assumed for the transition “exit.” All instances of the transition belong to the same set. In other words: if the transition “exit” is constantly enabled, it must be fire infinitely often.

The sets φ_x: Weak fairness must be assumed for the transitions “enter” and “test” for each value of x. In other words: if the process x ($1 \leq x \leq n$) is constantly able to execute either “enter” or “test,” it must do so infinitely often.

Fairness (3/3)

It is natural to define fairness assumptions directly for transition sets. MARIA is the first tool for high-level nets where this can be done. Traditionally fairness assumptions have been incorporated in the LTL formula, for instance

$$((\Diamond\Box e_1 \rightarrow \Box\Diamond o_1) \vee \cdots \vee (\Diamond\Box e_n \rightarrow \Box\Diamond o_n)) \rightarrow \Box(\text{pending}(x) \rightarrow \Diamond\text{critical}(x)).$$

This is utterly inefficient for two reasons:

1. In the worst case, the Büchi automaton for the LTL formula has an exponential number of states and transitions with regard to the number of temporal connectives.
2. The model must often be augmented: extra places (and a large number of new states) may be needed in order
   - to capture the firing of a transition with a state proposition, and
   - to model a scheduler.

On safety properties

Liveness properties are only meaningful when the executions are infinitely long, that is, the reachability graph of the system contains cycles or is infinite.

It is much easier to model check safety properties without Büchi automata. Whenever a new state is found, it is possible to check whether it fulfils the requirements.

Sometimes, it is possible to exploit the error handling routines of the modelling language. For instance, when modelling the TicketME algorithm with MARIA, the capacity of the place “critical” can be set to 0..1, so that an error will occur when the mutual exclusion property is violated.

LTL can express more complicated safety properties than $\Box p$. The safety subset of LTL can be translated into regular finite automata, which can be model checked by observing the states only, without detecting loops or considering fairness.