Chapter 2

Linear Temporal Logic

Reactive systems are characterized by a *continuous interaction* with the environment. For instance, an operating system or a coffee machine interacts with its environment (i.e. the user) and usually performs some actions, such as fetching a file or producing coffee. Thus a reactive system reacts to a stimulus from the environment by a reaction. After an interaction and accompanying reaction, a reactive system is — once more — able to interact. The continuous character of interaction in reactive systems differs from the traditional view on sequential systems. These sequential systems are considered as functions from inputs to outputs. After receiving input, the sequential system generates output(s) and terminates eventually after a finite number of computation steps. Due to the "transformation" of inputs into outputs these type of systems are sometimes also referred to as transformational systems. Properties of such transformational systems typically relate outputs generated to inputs provided in terms of pre- and postconditions as discussed previously.

Typical reactive systems are non-terminating. Due to their non-terminating behavior, properties of reactive systems we usually refer to the relative order of events in the system: i.e. the behavior of the system while over time. Lamport (1977) argued that the requirements that designers wish to impose on reactive systems fall basically into two categories:

- *Safety* properties state that "something bad never happens". A system satisfies such a property if it does not engage in the proscribed activity. For instance, a typical safety property for a coffee-machine is that the machine will provide never tea if the user requests coffee.

- *Liveness* properties state that "something good will eventually happen"; to satisfy such a property, the system must engage in some desired activity. An example liveness property for the coffee-machine is that it will provide coffee eventually after a sufficient payment by the user. Stated differently, the user will always be provided coffee at some time after inserting the payment.

Although informal, this classification has proven to be rather useful: the two classes of properties are (almost) disjoint, and most properties can be described as a combination of safety and liveness properties. (Later several authors refined and extended this classification with several other types of properties.)

Logics have been defined in order to describe precisely these type of properties. The most widely studied are *temporal logics*, which were introduced for specification and verification purposes in computer science by Pnueli (1977). (The origins of linear temporal logic go back to the field of philosophy where A. Prior invented it in the 1960s.) Temporal logics support the formulation of properties of system behavior over time. Different interpretations of temporal logics exist depending on how one considers the system to change with time.

1. *Linear* temporal logic allows the statement of properties of execution sequences of a system.

2. *Branching* temporal logic allow the user to write formulas which include some sensitivity to the choices available to a system during its execution. It allows the statement of properties of about possible execution sequences that start in a state.

In this chapter we deal with linear temporal logic; branching temporal logic will be treated in Chapter 3. Chapter 3 also contains a more extensive discussion about the differences between these two interpretations of temporal logic.

2.1 Syntax of PLTL

The syntax of propositional linear temporal logic is defined as follows. The starting point is a set of *atomic propositions*, i.e. propositions that cannot be further refined. Atomic propositions are the most basic statements that can be made. The set of atomic propositions is denoted by AP, and typical elements of AP are p, q and r. Examples of atomic propositions are x is greater than 0, or x equals one, given some variable x. Other examples are "it is raining" or "there are currently no customers in the shop". In principle, atomic propositions are defined over a set of variables $x, y, \ldots$, constants $0, 1, 2, \ldots$, functions $max, gcd, \ldots$ and predicates $x = 2, x \ mod \ 2 = 0, \ldots$, allowing e.g. statements such as $max(x, y) \leqslant 3$ or $x = y$. We will not dwell upon the precise definition of AP here and simple postulate its existence. AP is ranged over by p, q and r.

Notice that the choice of the set of atomic propositions AP is an important one; it fixes the most basic propositions that can be stated about the system under investigation. Fixing the set AP can therefore already be regarded as a *first step of abstraction*. If one, for instance, decides not to allow some system variables to be referred to in AP, then no property can be stated that refers to these variables, and consequently no such property can be checked.

The following definition determines the set of basic formulas that can be stated in propositional linear temporal logic (PLTL, for short).

Definition 1. (**Syntax of propositional linear temporal logic**)
Let AP be a set of atomic propositions. Then:

1. For all $p \in AP$, p is a formula.

2. If ϕ is a formula, then $\neg \phi$ is a formula.

3. If ϕ and ψ are formulas, then $\phi \vee \psi$ is a formula.
4. If ϕ is a formula, then $\mathsf{X}\,\phi$ is a formula.
5. If ϕ and ψ are formulas, then $\phi\,\mathsf{U}\,\psi$ is a formula.

The set of formulas constructed according to these rules is denoted by PLTL.

Note that the set of formulas obtained by the first three items gives us the set of formulas of propositional logic. Propositional logic is thus a proper subset of PLTL. The only temporal operators are X (pronounced "neXt") and U (pronounced "Until").

Alternatively, the syntax of PLTL can be given in Backus-Naur Form (BNF, for short) as follows: for $p \in AP$ the set of PLTL-formulas is defined by

$$\boxed{\phi ::= p \;\big|\; \neg\,\phi \;\big|\; \phi \vee \phi \;\big|\; \mathsf{X}\,\phi \;\big|\; \phi\,\mathsf{U}\,\phi.}$$

We will use this succinct notation for syntax definitions in the rest of these lecture notes.

The usual boolean operators $\wedge$ (conjunction), $\Rightarrow$ (implication) and $\Leftrightarrow$ (equivalence) are defined as

$$\begin{aligned} \phi \wedge \psi &\equiv \neg\,(\neg\,\phi \vee \neg\,\psi) \\ \phi \Rightarrow \psi &\equiv \neg\,\phi \vee \psi \\ \phi \Leftrightarrow \psi &\equiv (\phi \Rightarrow \psi) \wedge (\psi \Rightarrow \phi). \end{aligned}$$

The formula true equals $\phi \vee \neg\,\phi$ and false equals $\neg$ true. The temporal operators G (pronounced "always" or "Globally") and F (pronounced "eventually" or "Future") are defined by

$$\begin{aligned} \mathsf{F}\,\phi &\equiv \text{true}\,\mathsf{U}\,\phi \\ \mathsf{G}\,\phi &\equiv \neg\,\mathsf{F}\,\neg\,\phi. \end{aligned}$$

Since true is valid in all states, $\mathsf{F}\,\phi$ indeed denotes that ϕ holds at some point in the future. Suppose there is no point in the future for which $\neg\,\phi$ holds. Then, the counterpart of ϕ holds at any point. This explains the definition of $\mathsf{G}\,\phi$. In the literature F is sometimes denoted as $\Diamond$ and G as $\Box$. In these lecture notes we will use the traditional notations F and G.

To summarize, a formula without a temporal operator (X, F, G, U) at the "top level" refers to the current state, the formula $\mathsf{X}\,\phi$ to the next state, $\mathsf{G}\,\phi$ to all future states, $\mathsf{F}\,\phi$ to some future state, and U to all future states until a certain condition becomes valid.

In order to diminish the use of brackets, a precedence order on the operators is imposed as follows. As usual the unary operators bind stronger than the binary ones. For instance, we write $\neg\,\phi\,\mathsf{U}\,\mathsf{F}\,\psi$ instead of $(\neg\,\phi)\,\mathsf{U}\,(\mathsf{F}\,\psi)$. $\neg$ and X bind equally strong and bind stronger than F and G which also bind equally strong. The temporal operator U takes precedence over $\wedge$, $\vee$, and $\Rightarrow$. $\Rightarrow$ binds weaker than $\wedge$ and $\vee$, and $\wedge$ and $\vee$ bind equally strong. For instance,

$$((\neg\,\phi) \Rightarrow \psi)\,\mathsf{U}\,((\mathsf{X}\,\phi) \;\wedge\; (\mathsf{F}\,\psi))$$

is denoted by

$$(\neg\,\phi \Rightarrow \psi)\,\mathsf{U}\,(\mathsf{X}\,\phi \;\wedge\; \mathsf{F}\,\psi).$$

Example 1. *Let $AP = \{\, x = 1, x < 2, x \geqslant 3 \,\}$ be the set of atomic propositions. Example* PLTL*-formulas are:* $\mathsf{X}\,(x = 1)$, $\neg\,(x < 2)$, $x < 2 \;\vee\; x = 1$, $(x < 2)\,\mathsf{U}\,(x \geqslant 3)$, $\mathsf{F}\,(x < 2)$, *and* $\mathsf{G}\,(x = 1)$. *The second and the third formula are also propositional formulas. An example of a* PLTL*-formula in which temporal operators are nested is* $\mathsf{G}\,[(x < 2)\,\mathsf{U}\,(x \geqslant 3)]$. *(End of example.)*

2.2 Semantics of PLTL

The above definition provides us a recipe for the construction of PLTL-formulas, but it does not give an interpretation to these operators. Formally, PLTL is interpreted on sequences of states. Intuitively, $\mathsf{X}\,\phi$ means that ϕ is valid in the next state, $\mathsf{F}\,\phi$ means that ϕ is valid eventually, that is, it is valid now, or at some point in the future. But what do we mean by "state", "next state" and "some point in the future"? In order to define these notions unambiguously, a formal interpretation — usually called semantics — is given. The formal meaning of temporal logic formulas is defined in terms of a *model.*

Definition 2. (**Model for** PLTL)
A PLTL-model is a triple $\mathcal{M} = (S, R, \mathit{Label})$ where

- S is a non-empty denumerable set of states,
- $R : S \longrightarrow S$, assigning to $s \in S$ its unique successor state $R(s)$,
- $\mathit{Label} : S \longrightarrow 2^{AP}$, assigning to each state $s \in S$ the atomic propositions $\mathit{Label}(s)$ that are valid in s.

For the state $s \in S$, the state $R(s)$ is the unique next state of s. The important characteristic of the function R is that it acts as a generator for infinite sequences of states such as $s, R(s), R(R(s)), R(R(R(s))), \ldots$. For the semantics of PLTL these sequences of states are the cornerstones. One can equally well define a PLTL-model as a structure $(S, \sigma, \mathit{Label})$ where σ is an infinite sequence of states and S and Label are defined as above. In this alternative setting a finite sequence $s_0\, s_1\, \ldots\, s_k$ is identified with the infinite sequence $s_0\, s_1\, \ldots\, s_k\, s_k\, s_k\, \ldots$. This alternative approach is frequently encountered in the literature.

The function *Label* indicates which atomic propositions are valid for any state in $\mathcal{M}$. If for state s we have $\mathit{Label}(s) = \varnothing$ it means that no proposition is valid in s. The state s for which the proposition p is valid, i.e. $p \in \mathit{Label}(s)$, is sometimes referred to as a p-state.

Example 2. *Let $AP = \{ x = 0, x = 1, x \neq 0 \}$ be a set of atomic propositions, $S = \{ s_0, \ldots, s_3 \}$ a set of states, $R(s_i) = s_{i+1}$ for $0 \leqslant i < 3$ and $R(s_3) = s_3$ the successor function, and $Label(s_0) = \{ x \neq 0 \}$, $Label(s_1) = Label(s_2) = \{ x = 0 \}$, $Label(s_3) = \{ x = 1, x \neq 0 \}$ the function which assigns atomic propositions to states. In the model $\mathcal{M} = (S, R, Label)$ the atomic proposition "$x = 0$" is valid in states s_1 and s_2, "$x \neq 0$" is valid in s_0 and s_3, and "$x = 1$" is valid in state s_3 only.* *(End of example.)*

Note that an atomic proposition can either hold or not hold in any state of $\mathcal{M}$. No further interpretation is given. For instance, if "$x = 1$" is valid in state s, it does not mean that "$x \neq 0$" is also valid in that state. Technically this results from the fact that no constraints are put on the labelling *Label* of states with atomic propositions.

The meaning of formulas in logic is defined by means of a satisfaction relation (denoted by $\models$) between a model $\mathcal{M}$, one of its states s, and a formula ϕ. $(\mathcal{M}, s, \phi) \in \models$ is denoted by the following infix notation: $\mathcal{M}, s \models \phi$. The concept is that $\mathcal{M}, s \models \phi$ if and only if ϕ is valid in the state s of the model $\mathcal{M}$. When the model $\mathcal{M}$ is clear from the context we often omit the model and simply write $s \models \phi$ rather than $\mathcal{M}, s \models \phi$.

Definition 3. (**Semantics of** PLTL)
Let $p \in AP$ be an atomic proposition, $\mathcal{M} = (S, R, Label)$ a PLTL-model, $s \in S$, and ϕ, ψ PLTL-formulas. The satisfaction relation $\models$ is defined by:

$$
\begin{array}{ll}
s \models p & \text{iff } p \in Label(s) \\
s \models \neg\, \phi & \text{iff } \neg\,(s \models \phi) \\
s \models \phi \vee \psi & \text{iff } (s \models \phi) \vee (s \models \psi) \\
s \models \mathsf{X}\, \phi & \text{iff } R(s) \models \phi \\
s \models \phi \,\mathsf{U}\, \psi & \text{iff } \exists\, j \geqslant 0.\, R^j(s) \models \psi \ \wedge\ (\forall\, 0 \leqslant k < j.\, R^k(s) \models \phi).
\end{array}
$$

Here $R^0(s) = s$, and $R^{n+1}(s) = R^n(R(s))$ for any $n \geqslant 0$.

(Although the logical operators on the right-hand side of a defining equation are denoted in the same way as the logical operators in the logic, they represent operators at a meta-level.) If $R(s) = s'$, the state s' is called a direct successor of s. If $R^n(s) = s'$ for $n \geqslant 1$, state s' is called a successor of s. If $\mathcal{M}, s \models \phi$ we say that model $\mathcal{M}$ satisfies ϕ in state s. Stated otherwise, the formula ϕ holds in state s of model $\mathcal{M}$.

The formal interpretation of the other connectives true, false, $\wedge$, $\Rightarrow$, G, and F can now be obtained from this definition by straightforward calculation. It is not difficult to see that true is valid in all states: take true $\equiv p \vee \neg\, p$, and the validity of true in state s reduces to $p \in Label(s) \vee p \notin Label(s)$ which is a valid statement for all states s. So $\mathcal{M}, s \models$ true for all $s \in S$. As an example we derive for the semantics of $\mathsf{F}\, \phi$:

$$
\begin{array}{ll}
& s \models \mathsf{F}\, \phi \\
\Leftrightarrow & \{ \text{ definition of } \mathsf{F} \ \} \\
& s \models \text{true} \,\mathsf{U}\, \phi \\
\Leftrightarrow & \{ \text{ semantics of } \mathsf{U} \ \} \\
& \exists\, j \geqslant 0.\, R^j(s) \models \phi \ \wedge\ (\forall\, 0 \leqslant k < j.\, R^k(s) \models \text{true})
\end{array}
$$

$$
\begin{array}{ll}
\Leftrightarrow & \{ \text{ calculus } \} \\
 & \exists\, j \geqslant 0.\, R^j(s) \models \phi.
\end{array}
$$

Therefore, $\mathsf{F}\,\phi$ is valid in s if and only if there is some (not necessarily direct) successor state of s, or s itself where ϕ is valid.

Using this result for F, for the semantics of $\mathsf{G}\,\phi$ we now become:

$$
\begin{array}{ll}
 & s \models \mathsf{G}\,\phi \\
\Leftrightarrow & \{ \text{ definition of } \mathsf{G} \ \} \\
 & s \models \neg\,\mathsf{F}\,\neg\,\phi \\
\Leftrightarrow & \{ \text{ using result of previous derivation } \} \\
 & \neg\,[\exists\, j \geqslant 0.\, R^j(s) \models \neg\,\phi] \\
\Leftrightarrow & \{ \text{ semantics of } \neg\,\phi \ \} \\
 & \neg\,[\exists\, j \geqslant 0.\, \neg\,(R^j(s) \models \phi)] \\
\Leftrightarrow & \{ \text{ predicate calculus } \} \\
 & \forall\, j \geqslant 0.\, R^j(s) \models \phi.
\end{array}
$$

Therefore, the formula $\mathsf{G}\,\phi$ is valid in s if and only if for all successor states of s, including s itself, the formula ϕ is valid. This explains the expression "always" ϕ.

Example 3. *If the formula $\phi\,\mathsf{U}\,\psi$ is valid in s, then $\mathsf{F}\,\psi$ is also valid in s. Thus when stating $\phi\,\mathsf{U}\,\psi$ it is implicitly assumed that there exists some future state for which ψ holds. A weaker variant of "until", the "unless" operator W, states that ϕ holds continuously either until ψ holds for the first time, or throughout the sequence. The unless operator W is defined by*

$$\phi\,\mathsf{W}\,\psi \;\equiv\; \mathsf{G}\,\phi \;\vee\; (\phi\,\mathsf{U}\,\psi).$$

(End of example.)

Example 4. *Let $\mathcal{M}$ be given as a sequence of states depicted as the first row in Figure 2.1. States are depicted as circles and the function R is denoted by arrows, i.e. there is an arrow from s to s' iff $R(s) = s'$. Since R is a (total) function, each state has precisely one outgoing arrow. The labelling* Label *is indicated below the states.*

In the lower rows the validity of the three formulas is shown, $\mathsf{F}\,p$, $\mathsf{G}\,p$, and $q\,\mathsf{U}\,p$, for all states in the given model $\mathcal{M}$. A state is colored black if the formula is valid in that state, and colored white otherwise. The formula $\mathsf{F}\,p$ is valid in all but the last state. No p-state can be reached from the last state. $\mathsf{G}\,p$ is invalid in all states, since there is no state that only has p-states as successors. From all q-states the (unique) p-state can be reached in zero or more steps. Therefore, for all these states the formula $q\,\mathsf{U}\,p$ holds. *(End of example.)*

Example 5. *Let $\mathcal{M}$ be given as the sequence of states depicted in Figure 2.2 where p through t are atomic propositions. Consider the formula $\mathsf{X}\,[r \;\Rightarrow\; (q\,\mathsf{U}\,s)]$ which equals $\mathsf{X}\,[\neg\,r \;\vee\; (q\,\mathsf{U}\,s)]$. For all states whose direct successor does not satisfy r, the formula is valid due to the first disjunct. This applies to the second state. In the third state the formula is not valid since r*

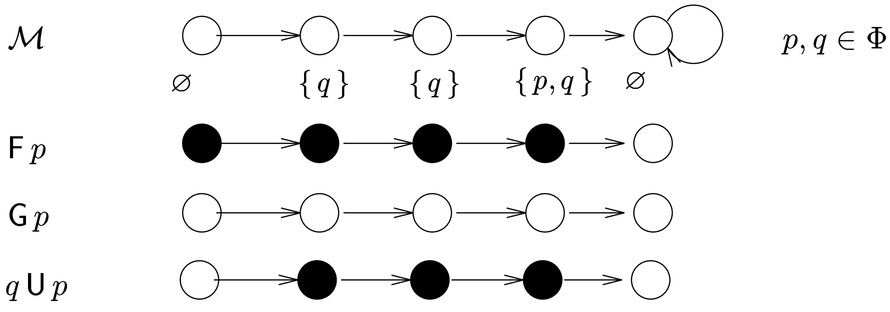


Figure 2.1: Example of interpretation of PLTL-formulas (I)

is valid in its successor state but q and s are not. Finally, in the first and the fourth state the formula is valid, since in their direct successor state q is valid and in the state thereafter s is, so $q\, \mathsf{U}\, s$ is valid in their direct successor states. The validity of the formulas $\mathsf{G}\, p$, $\mathsf{F}\, t$, $\mathsf{G}\,\mathsf{F}\, s$ and the formula $\mathsf{X}\, [r \;\Rightarrow\; (q\, \mathsf{U}\, s)]$ is indicated in the rows depicted below $\mathcal{M}$. (End of example.)

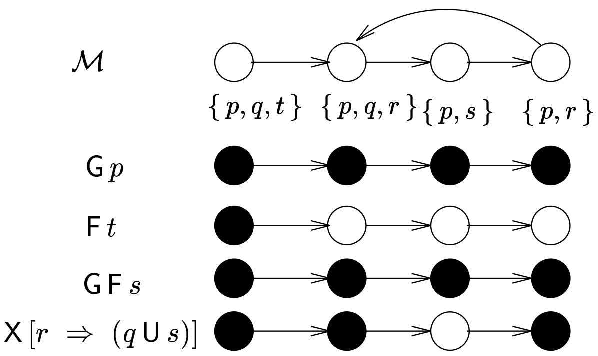


Figure 2.2: Example of interpretation of PLTL-formulas (II)

We give below some frequently used propositions in PLTL and provide their verbal interpretation such that $\mathcal{M}, s \models \phi$, i.e. ϕ holds in state s of $\mathcal{M}$.

1. $\phi \;\Rightarrow\; \mathsf{F}\, \psi$: if initially (that is, in s) ϕ, then eventually ψ.
2. $\mathsf{G}\, [\phi \;\Rightarrow\; \mathsf{F}\, \psi]$: if ϕ then eventually ψ (for all successor states of s and the initial state s).
3. $\mathsf{G}\,\mathsf{F}\, \phi$: ϕ holds infinitely often.
4. $\mathsf{F}\,\mathsf{G}\, \phi$: eventually permanently ϕ.
5. $\mathsf{G}\, [\phi \;\Rightarrow\; \mathsf{X}\, \phi]$: if ϕ is valid in some state, then it is also valid in its successor state.
6. $\mathsf{G}\, [\phi \;\Rightarrow\; \mathsf{G}\, \phi]$: once ϕ, always ϕ.

Model checking, satisfiability and validity

In the first chapter we gave an informal definition of the model checking problem. Given the formal machinery developed so far we are now in a position to give a more precise characterisation. The model checking problem can formally be defined as:

> *The model checking problem is: given a (finite) model $\mathcal{M}$, a state s and a property ϕ, do we have $\mathcal{M}, s \models \phi$?*

The model checking problem should not be confused with the more traditional satisfiability problem in logic.

The *satisfiability* problem can be phrased as follows: given a property ϕ, does there exist a model $\mathcal{M}$ and state s, such that $\mathcal{M}, s \models \phi$? While for model checking the model $\mathcal{M}$ (and state s) is given, this is not the case for the satisfiability problem. For PLTL the satisfiability problem is decidable. Since satisfiability is decidable, the (simpler) model checking problem for PLTL is also decidable. Satisfiability is relevant to system validation using formal verification (cf. Chapter 1) in the following sense. As an example consider a system specification and its implementation, both specifications being formalised as PLTL-formulas ϕ and ψ. Checking whether the implementation conforms to the specification now reduces to checking whether $\psi \Rightarrow \phi$. By determining the satisfiability of this formula one is able to check whether there exists a model for which this desired relation does exist. If the formula is not satisfiable, that is, no such model does exist, it means that the relation between the implementation and specification cannot be realised in any model. Thus, the implementation is not a correct implementation of the specification.

A related problem that occurs frequently in the literature is the *validity* problem: given a property ϕ, do we have for all models $\mathcal{M}$ and all states s in these models, $\mathcal{M}, s \models \phi$? The difference to the satisfiability problem is that to solve the validity problem one has to check whether $\mathcal{M}, s \models \phi$ for *all* existing models $\mathcal{M}$ and states s, rather than to determine the existence of one (or more) such $\mathcal{M}$ and s. Logically speaking, the validity problem for ϕ equals the negation of the satisfiability problem for $\neg\,\phi$.

(Aside: at first sight the validity problem seems to be undecidable — there does not seem to be an effective method for deciding whether formula ϕ is valid in *all* states of *all* models — since PLTL-models are allowed to have an infinite state space. However, due to the so-called finite-model property for PLTL, it suffices to consider only all models with a finite number of states to check validity. The finite-model property states that if a PLTL-formula is satisfiable, then it is satisfiable in a model with a finite set of states.)

2.3 Axiomatization

The validity of a PLTL-formula can of course be derived using the semantics as given before. This is usually a rather cumbersome task since we have to reason about the formal semantics which is defined in terms of the model $\mathcal{M}$. Let us, for instance, try to deduce that

$$\phi \,\mathsf{U}\, \psi \;\Leftrightarrow\; \psi \,\vee\, [\phi \,\wedge\, \mathsf{X}\,(\phi \,\mathsf{U}\, \psi)].$$

Intuitively this equation is valid: if in the current state ψ holds, then obviously $\phi \,\mathsf{U}\, \psi$ holds (for arbitrary ϕ), since ψ can be reached via a path of length 0. Otherwise if ϕ holds in the current state and in the next state $\phi \,\mathsf{U}\, \psi$ holds, then $\phi \,\mathsf{U}\, \psi$ holds. We obtain the following derivation:

$$
\begin{array}{ll}
 & s \models \psi \;\vee\; [\phi \;\wedge\; \mathsf{X}\,(\phi \,\mathsf{U}\, \psi)] \\
\Leftrightarrow & \{ \text{ semantics of } \wedge \text{ and } \vee \text{ } \} \\
 & (s \models \psi) \;\vee\; (s \models \phi \;\wedge\; s \models \mathsf{X}\,(\phi \,\mathsf{U}\, \psi)) \\
\Leftrightarrow & \{ \text{ semantics of } \mathsf{X} \text{ } \} \\
 & (s \models \psi) \;\vee\; (s \models \phi \;\wedge\; R(s) \models \phi \,\mathsf{U}\, \psi) \\
\Leftrightarrow & \{ \text{ semantics of } \mathsf{U} \text{ } \} \\
 & \quad (s \models \psi) \\
 & \vee\; (s \models \phi \;\wedge\; [\exists\, j \geqslant 0.\, R^j(R(s)) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j.\, R^k(R(s)) \models \phi]) \\
\Leftrightarrow & \{ \text{ calculus using } R^{n+1}(s) = R^n(R(s)) \text{ } \} \\
 & \quad (s \models \psi) \\
 & \vee\; [\exists\, j \geqslant 0.\, R^{j+1}(s) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j.\, (R^{k+1}(s) \models \phi \;\wedge\; s \models \phi)] \\
\Leftrightarrow & \{ \text{ calculus using } R^0(s) = s \text{ } \} \\
 & \quad (s \models \psi) \;\vee\; [\exists\, j \geqslant 0.\, R^{j+1}(s) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j{+}1.\, R^k(s) \models \phi] \\
\Leftrightarrow & \{ \text{ calculus using } R^0(s) = s \text{ } \} \\
 & \quad [\exists\, j = 0.\, R^0(s) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j.\, R^k(s) \models \phi] \\
 & \vee\; [\exists\, j \geqslant 0.\, R^{j+1}(s) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j{+}1.\, R^k(s) \models \phi] \\
\Leftrightarrow & \{ \text{ predicate calculus } \} \\
 & [\exists\, j \geqslant 0.\, R^j(s) \models \psi \;\wedge\; \forall\, 0 \leqslant k < j.\, R^k(s) \models \phi] \\
\Leftrightarrow & \{ \text{ semantics of } \mathsf{U} \text{ } \} \\
 & s \models \phi \,\mathsf{U}\, \psi.
\end{array}
$$

As we can see from this calculation, formula manipulation is tedious and error-prone. A more effective way to check the validity of formulas is to use the syntax of the formulas rather than their semantics. The concept is to define a set of proof rules that allow the rewriting of PLTL-formulas into semantically equivalent PLTL-formulas at a syntactical level. If, for instance, $\phi \equiv \psi$ is a rule (axiom), then $s \models \phi$ if and only if $s \models \psi$ for all $\mathcal{M}$ and state s. Semantically equivalent means that for all states in all models these rules are valid. The set of proof rules obtained in this way is called an *axiomatization*.

It is not the aim of this section to deal with all possible proof rules for PLTL but rather to give the reader some basic rules that are convenient. The rules presented below are grouped and each group has been given a name, for reference purposes. Using the idempotency and the absorption rules any non-empty sequence of F and G can be reduced to either F, G, $\mathsf{F}\,\mathsf{G}$, or $\mathsf{G}\,\mathsf{F}$. The validity of these proof rules can be proven using the semantic interpretation as we have seen for the first expansion law above. The difference is that we only have to perform these tedious proofs once; thereafter these rules can be universally applied. Note that the validity of the expansion rules for F and G follows directly from the validity of the expansion rule for U, using the definition of F and G.

Duality rules:	$\neg\,\mathsf{G}\,\phi$	$\equiv$	$\mathsf{F}\,\neg\,\phi$
	$\neg\,\mathsf{F}\,\phi$	$\equiv$	$\mathsf{G}\,\neg\,\phi$
	$\neg\,\mathsf{X}\,\phi$	$\equiv$	$\mathsf{X}\,\neg\,\phi$
Idempotency rules:	$\mathsf{G}\,\mathsf{G}\,\phi$	$\equiv$	$\mathsf{G}\,\phi$
	$\mathsf{F}\,\mathsf{F}\,\phi$	$\equiv$	$\mathsf{F}\,\phi$
	$\phi\,\mathsf{U}\,(\phi\,\mathsf{U}\,\psi)$	$\equiv$	$\phi\,\mathsf{U}\,\psi$
	$(\phi\,\mathsf{U}\,\psi)\,\mathsf{U}\,\psi$	$\equiv$	$\phi\,\mathsf{U}\,\psi$
Absorption rules:	$\mathsf{F}\,\mathsf{G}\,\mathsf{F}\,\phi$	$\equiv$	$\mathsf{G}\,\mathsf{F}\,\phi$
	$\mathsf{G}\,\mathsf{F}\,\mathsf{G}\,\phi$	$\equiv$	$\mathsf{F}\,\mathsf{G}\,\phi$
Commutation rule:	$\mathsf{X}\,(\phi\,\mathsf{U}\,\psi)$	$\equiv$	$(\mathsf{X}\,\phi)\,\mathsf{U}\,(\mathsf{X}\,\psi)$
Expansion rules:	$\phi\,\mathsf{U}\,\psi$	$\equiv$	$\psi \,\vee\, [\phi \,\wedge\, \mathsf{X}\,(\phi\,\mathsf{U}\,\psi)]$
	$\mathsf{F}\,\phi$	$\equiv$	$\phi \,\vee\, \mathsf{X}\,\mathsf{F}\,\phi$
	$\mathsf{G}\,\phi$	$\equiv$	$\phi \,\wedge\, \mathsf{X}\,\mathsf{G}\,\phi$

An equational rule is said to be *sound* if it is valid. Formally, the axiom $\phi \equiv \psi$ is called sound if and only if, for any PLTL-model $\mathcal{M}$ and state s in $\mathcal{M}$:

$\mathcal{M}, s \models \phi$ if and only if $\mathcal{M}, s \models \psi$

Applying sound axioms to a certain formula means that the validity of that formula is unchanged: the axioms do not change the semantics of the formula at hand. If for any semantically equivalent ϕ and ψ it is possible to derive this equivalence using axioms then the axiomatization is said to be *complete*. The list of axioms given before for PLTL is sound, but not complete. A sound and complete axiomatization for PLTL does exist, but falls outside the scope of these lecture notes.

2.4 Extensions of PLTL (optional)

Strict and non-strict interpretation. $\mathsf{G}\,\phi$ means that ϕ holds at all states including the current state. It is called a *non-strict* interpretation since it also refers to the current state. Opposed to this a *strict* interpretation does not refer to the current state. The strict interpretation of G, denoted $\tilde{\mathsf{G}}$, is defined by $\tilde{\mathsf{G}}\,\phi \equiv \mathsf{X}\,\mathsf{G}\,\phi$. That is, $\tilde{\mathsf{G}}\,\phi$ means that ϕ holds at all successor states without stating anything about the current state. Similarly, we have the strict variants of F and U that are defined by $\tilde{\mathsf{F}}\,\phi \equiv \mathsf{X}\,\mathsf{F}\,\phi$ and $\phi\,\tilde{\mathsf{U}}\,\psi \equiv \mathsf{X}\,(\phi\,\mathsf{U}\,\psi)$. (Notice that for X it does not make much sense to distinguish between a strict and a non-strict interpretation.) These definitions show that the strict interpretation can be defined in terms of the non-strict interpretation. In the opposite direction we have:

$$\mathsf{G}\,\phi \;\equiv\; \phi \,\wedge\, \tilde{\mathsf{G}}\,\phi$$

$$\begin{array}{rcl} \mathsf{F}\,\phi & \equiv & \phi \,\wedge\, \tilde{\mathsf{F}}\,\phi \\ \phi\,\mathsf{U}\,\psi & \equiv & \psi \,\vee\, [\phi \,\wedge\, (\phi\,\tilde{\mathsf{U}}\,\psi)]. \end{array}$$

The first two equations are self-explanatory given the above definitions of $\tilde{\mathsf{G}}$ and $\tilde{\mathsf{F}}$. The third equation is justified by the expansion rule of the previous section: if we substitute $\phi\,\tilde{\mathsf{U}}\,\psi \equiv \mathsf{X}\,(\phi\,\mathsf{U}\,\psi)$ in the third equation we indeed obtain the expansion rule for U. Although sometimes the strict interpretation is used, it is more common to use the non-strict interpretation as is done in these lecture notes.

Past operators. All operators refer to the future including the current state. Therefore, the operators are known as future operators. Sometimes PLTL is extended with *past* operators. This can be useful for a particular application, since sometimes properties are more easily expressed in terms of the past than in terms of the future. For instance, $\underline{\mathsf{G}}\,\phi$ ("always in the past") means — in the non-strict interpretation — that ϕ is valid now and in any state in the past. $\underline{\mathsf{F}}\,\phi$ ("sometime in the past") means that either ϕ is valid in the current state or in some state in the past and $\underline{\mathsf{X}}\,\phi$ means that ϕ holds in the previous state, if such state exists. As for future operators, also for past operators a strict and a non-strict interpretation can be given. The main reason for introducing past operators is to make it easier to write a specification; the expressive power of PLTL is not increased by the addition of past operators (Lichtenstein, Pnueli & Zuck, 1985)[1].

2.5 Specifying properties in PLTL

In order to give the reader some idea of how to formulate informal properties in linear temporal logic, we treat two examples. In the first example we try to formalize properties of a simple communication system, and in the second example we treat a formal requirement specification of a leader election protocol in a distributed system where processes can start at arbitrary moments.

A communication channel

Consider a unidirectional channel between two communicating processes: a sender (S) and a receiver (R). S is equipped with an output buffer (S.out) and R with an input buffer (R.in). Both buffers have an infinite capacity. If S sends a message m to R it inserts the message into its output buffer S.out. The output buffer S.out and the input buffer R.in are connected via a unidirectional channel. R receives messages by deleting messages from its input buffer R.in. Assume that all messages are uniquely identified, and assume the set of atomic propositions $AP = \{\, m \in S.out, m \in R.in \,\}$ where m denotes a message. We assume implicitly that all properties are stated for all messages m (i.e. universal quantification over m is assumed). This is for convenience and does not affect the decidability of the satisfiability problem if we assume that the number of messages is finite. In addition, we assume that the buffers S.out and R.in behave in a normal way, i.e., they do not disrupt or lose messages, and messages cannot stay in a buffer infinitely long.

[1]If as an underlying semantic notion a discrete model is taken, as in this chapter.

We formalize the following informal requirements on the channel in PLTL:

- "A message cannot be in both buffers at the same time".

$$\mathsf{G}\,\neg\,(m \in S.out \ \wedge\ m \in R.in)$$

- "The channel does not lose messages". If the channel does not lose messages it means that messages that are in S.out will eventually be delivered in R.in:

$$\mathsf{G}\,[m \in S.out \ \Rightarrow\ \mathsf{F}\,(m \in R.in)]$$

  (Notice that if we do not assume uniqueness of messages then this property does not suffice, since if e.g. two copies of m are transmitted and only the last copy is eventually received this would satisfy this property. It has been shown by several authors that uniqueness of messages is a prerequisite for the specification of requirements for message-passing systems in temporal logic.) Given the validity of the previous property, we can equally well state

$$\mathsf{G}\,[m \in S.out \ \Rightarrow\ \mathsf{X}\,\mathsf{F}\,(m \in R.in)]$$

  since m cannot be in S.out and R.in at the same time.

- "The channel is order-preserving". This means that if m is offered first by S to its output buffer S.out and subsequently m', then m will be received by R before m':

$$\begin{aligned}&\mathsf{G}\,[m \in S.out \ \wedge\ \neg\, m' \in S.out \ \wedge\ \mathsf{F}\,(m' \in S.out)\\ &\Rightarrow\ \mathsf{F}\,(m \in R.in \ \wedge\ \neg\, m' \in R.in \ \wedge\ \mathsf{F}\,(m' \in R.in))]\end{aligned}$$

  Notice that in the premise the conjunct $\neg\, m' \in S.out$ is needed in order to specify that m' is put in S.out after m. $\mathsf{F}\,(m' \in S.out)$ on its own does not exclude that m' is already in the sender's buffer when message m is in S.out. (It is left to the reader to investigate what the meaning is when the first and third occurrence of F is replaced by X.)

- "The channel does not spontaneously generate messages". This means that for any m in R.in, it must have been previously sent by S. Using the past operator $\underline{\mathsf{F}}$ this can be formalized conveniently by:

$$\mathsf{G}\,[m \in R.in \ \Rightarrow\ \underline{\mathsf{F}}\,(m \in S.out)]$$

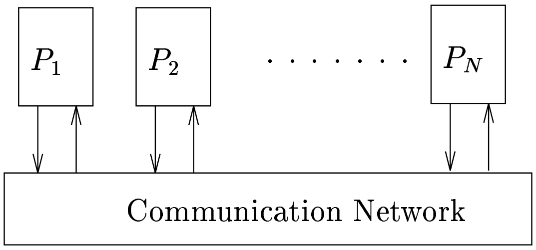
  In the absence of past operators, a it is possible to use the U-operator:

$$\mathsf{G}\,[(\neg\, m \in R.in)\,\mathsf{U}\,(m \in S.out)]$$

Dynamic leader election

In current distributed systems several functions (or services) are offered by some dedicated process(es) in the system. One might think of address assignment and registration, query co-ordination in a distributed database system, clock distribution, token regeneration after token loss in a token ring network, initiation of topology updates in a mobile network, load balancing, and so forth. Usually many processes in the system are potentially capable doing this. However, for consistency reasons at any time only one process is allowed to actually offer the function. Therefore, one process — called the "leader" — must be elected to support that function. Sometimes it suffices to elect an arbitrary process, but for other functions it is important to elect the process with the best capabilities for performing that function. Here we abstract from specific capabilities and use ranking on basis of process identities. The idea is thus that a process with a higher identity has better capabilities.

Assume we have a *finite* number of processes connected via some communication means. The communication between processes is asynchronous, as in the previous example. Pictorially,



Each process has a unique identity, and it is assumed that a total ordering exists on these identities. Processes behave dynamically in the sense that they are initially inactive (so, not participating in the election) and may become active (i.e., participating in the election) at arbitrary moments. In order to have some progress we assume that a process cannot be inactive indefinitely; i.e. each process becomes active at some time. Once a process participates it continues to do so; it does not become inactive anymore. For a given set of active processes a leader will be elected; if an inactive process becomes active a new election takes place if this process has a higher identity than the current leader.

Assume the set of atomic propositions

$$AP = \{ \, leader_i, active_i, i < j \mid 0 < i, j \leqslant N \, \}$$

where $leader_i$ means that process i is a leader, $active_i$ means that process i is active, and $i < j$ that the identity of i is smaller than the identity of j (in the total ordering on identities). We will use i, j as process identities. N is the number of processes. Assume that a process that is inactive is not a leader.

In the following formulations we use universal and existential quantifications over the set of identities. Strictly speaking, these are not part of PLTL. Since we deal with a finite number of processes the universal quantification $\forall \, i. \, P(i)$, where P is some proposition on processes, can be expanded into $P(1) \ \wedge \ \ldots \ \wedge \ P(N)$, and similarly we can expand $\exists i. \, P(i)$. The quantifications can thus be considered as simply abbreviations.

- "There is always one leader".

$$\mathsf{G}\,[\exists\, i.\, \mathit{leader}_i \;\wedge\; (\forall\, j \neq i.\, \neg\, \mathit{leader}_j)]$$

Since we deal with a dynamic system where all processes may be inactive initially (and thus no leader exists), this property will in general not be valid. Also, in a distributed system with asynchronous communication switching from one leader to another can hardly be made atomic, so it is convenient to allow the temporary absence of a leader. As a first attempt one could think of modifying the above formula into:

$$\mathsf{G}\,\mathsf{F}\,[\exists\, i.\, \mathit{leader}_i \;\wedge\; (\forall\, j \neq i.\, \neg\, \mathit{leader}_j)]$$

which allows there temporarily to be no leader, but which also allows there to be more than one leader at a time temporarily. For consistency reasons this is not desired. We therefore consider the following two properties.

- "There must always be at most one leader".

$$\mathsf{G}\,[\mathit{leader}_i \;\Rightarrow\; \forall\, j \neq i.\, \neg\, \mathit{leader}_j]$$

- "There will be enough leaders in due time". (This requirement avoids the construction of a leader election protocol that never elects a leader. Such a protocol would fulfill the previous requirement, but is not intended.)

$$\mathsf{G}\,\mathsf{F}\,[\exists\, i.\, \mathit{leader}_i]$$

This property does not imply that there will be infinitely many leaders. It only states that there are infinitely many states at which a leader exists.

- "In the presence of an active process with a higher identity the leader will resign at some time"

$$\mathsf{G}\,[\forall\, i, j.\; ((\mathit{leader}_i \;\wedge\; i < j \;\wedge\; \neg\, \mathit{leader}_j \;\wedge\; \mathit{active}_j) \;\Rightarrow\; \mathsf{F}\,\neg\, \mathit{leader}_i)]$$

For reasons of efficiency it is assumed not to be desirable that a leader eventually resigns in presence of an inactive process that may participate at some unknown time in the future. Therefore we require j to be an active process.

- "A new leader will be an improvement over the previous one". This property requires that successive leaders have an increasing identity.

$$\mathsf{G}\,[\forall\, i, j.\, (\mathit{leader}_i \;\wedge\; \neg\,\mathsf{X}\, \mathit{leader}_i \;\wedge\; \mathsf{X}\,\mathsf{F}\, \mathit{leader}_j) \;\Rightarrow\; i < j]$$

Note that this requirement implies that a process that resigns once, will not become a leader any more.

2.6 Summary of steps in PLTL-model checking

An overview of the different steps of model checking PLTL is shown in Figure 2.3. The model of the system sys is transformed into a Büchi automaton A_{sys} in which all states are accepting. The property to be checked is specified in PLTL — yielding ϕ —, negated, and subsequently transformed into a second Büchi automaton $A_{\neg\phi}$. The product of these two automata represents all possible computations that violate ϕ. By checking the emptiness of this automaton, it is thus determined whether ϕ is satisfied by the system-model sys or not.

Although the various steps in the algorithm are presented in a strict sequential order, that is, indicating that the next step cannot be performed, until the previous ones are completely finished, some steps can be done *on-the-fly*. For instance, constructing the graph for a normal-form formula can be performed while checking the emptiness of the product automaton $A_{sys} \otimes A_{\neg\phi}$. In this way the graph is constructed "on demand": only a new vertex is considered if no accepting cycle has been found yet in the partially constructed product automaton. When the successors of a vertex in the graph are constructed, one chooses the successors that match the current state of the automaton A_{sys} rather than all possible successors. Thus it is possible that an accepting cycle is found (i.e. a violation of ϕ with corresponding counter-example) without the need for generating the entire graph $\mathcal{G}_{\phi}$. In a similar way, the automaton A_{sys} does not need to be totally available before starting checking non-emptiness of the product automaton. This is usually the most beneficial step of on-the-fly model checking, since this automaton is typically rather large, and avoiding the entire consideration of this automaton may reduce the state space requirements significantly.

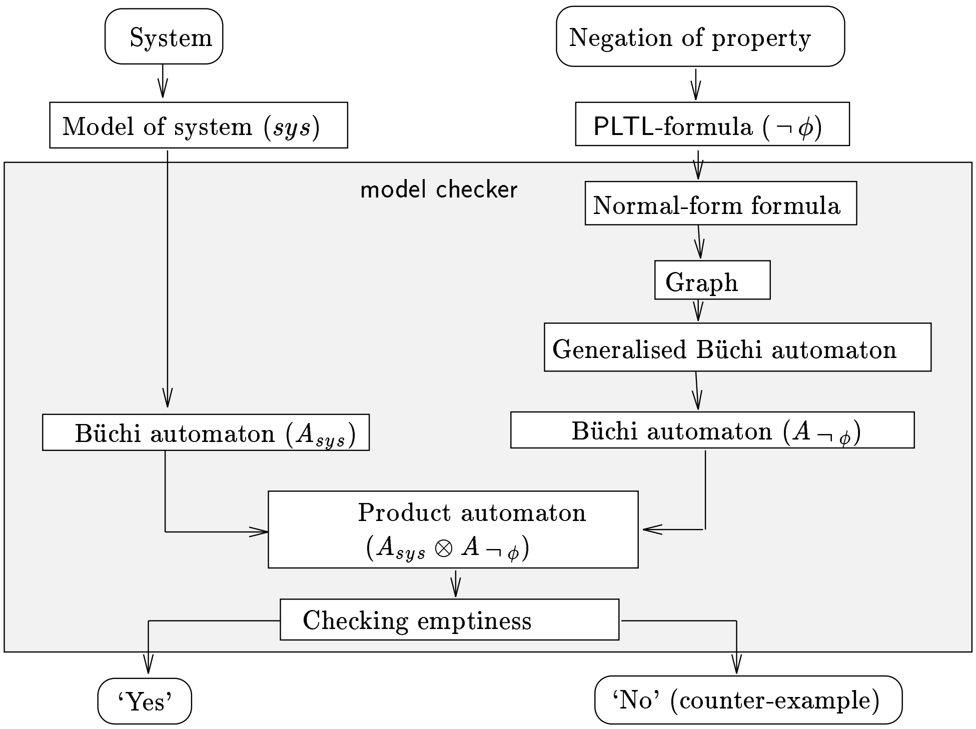


Figure 2.3: Overview of model-checking PLTL

We conclude by discussing the worst-case time complexity of the model-checking recipe for PLTL-formulas. Let ϕ be the formula to be checked and *sys* the model of the system under consideration. The crucial step is the transformation of a normal form PLTL-formula into a graph. Since each vertex in the graph is labelled with a set of sub-formulas of ϕ, the number of vertices in the graph is proportional to the number of sets of sub-formulas of ϕ, that is $\mathcal{O}(2^{|\phi|})$. Since the other steps of the transformation of ϕ into an LBA do not affect this worst-case complexity, the resulting LBA has a state space of $\mathcal{O}(2^{|\phi|})$. The worst-case space complexity of the product automaton $A_{sys} \otimes A_{\neg\phi}$ is therefore $\mathcal{O}(\mid S_{sys} \mid \times 2^{|\phi|})$ where S_{sys} denotes the set of states in the LBA A_{sys}. Since the time complexity of checking the emptiness of an LBA is linear in the number of states and transitions we obtain that

> *The worst-case time complexity of checking whether system-model sys satisfies the* PLTL*-formula* ϕ *is* $\mathcal{O}(\mid S_{sys} \mid^2 \times 2^{|\phi|})$

since in worst case we have $\mid S_{sys} \mid^2$ transitions. In the literature it is common to say that the time complexity of model checking PLTL is linear in the size of the model (rather than quadratic) and exponential in the size of the formula to be checked.

The fact that the time complexity is exponential in the length of the PLTL-formula seems to be a major obstacle for the application of this algorithm to practical systems. Experiments have shown that this dependency is not significant, since the length of the property to be checked is usually rather short. This is also justified by several industrial case studies. (Holzmann, for instance, declares that "PLTL-formulas have rarely more than 2 or 3 operators".)

We conclude by briefly indicating the space complexity of model checking PLTL. The model checking problem for PLTL is PSPACE-complete, i.e. a state space that is polynomial in the size of the model and the formula is needed (Sistla and Clarke, 1985).

2.7 Selected references

Linear temporal logic:

- A. Pnueli. The temporal logic of programs. *Proceedings 18th IEEE Symposium on Foundations of Computer Science*, pages 46–57, 1977.
- R. Gotzhein. Temporal logic and applications–a tutorial. *Computer Networks & ISDN Systems*, **24**: 203–218, 1992.
- Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: Specification.* Springer-Verlag, 1992.

Past operators in temporal logic:

- O. Lichtenstein, A. Pnueli, and L. Zuck. The glory of the past. *Logics of Programs*, LNCS 193, pages 196–218, 1985.

Büchi automata and the connection between PLTL and LBAs:

- M. VARDI. An automata-theoretic approach to linear temporal logic. *Logics for Concurrency – Structure versus Automata*, LNCS 1043, pages 238–265, 1995.
- P. WOLPER, M. VARDI, A.P. SISTLA. Reasoning about infinite computation paths. *Proceedings 24th IEEE Symposium on Foundations of Computer Science*, pages 185–194, 1983.
- E.A. EMERSON AND C.-L. LEI. Modalities for model checking: branching time logic strikes back. *Proceedings 12th ACM Symposium on Principles of Programming Languages*, pages 84–96, 1985.
- R. GERTH, D. PELED, M. VARDI, P. WOLPER. Simple on-the-fly automatic verification of linear temporal logic. *Proceedings 13th Symposium on Protocol Specification, Testing and Verification*, pages 3–18, 1995.

Complexity of PLTL:

- A.P. SISTLA AND E.M. CLARKE. The complexity of propositional linear temporal logics. *Journal of the ACM* **32**(3): 733–749, 1985.