

Fondements pour la vérification des systèmes temps-réel et concurrents

Lecture 1

Introduction to formal languages for specifications

Stéphane Demri

September 24, 2007

Structure du cours

- ▶ Logiques temporelles
 - ▶ Intervenants: S. Demri & N. Markey
 - ▶ Volume horaire: 15h

- ▶ Automates temporisés
 - ▶ Intervenants: F. Laroussinie & N. Markey
 - ▶ Volume horaire: 15h

- ▶ Modélisation et vérification de systèmes distribués
 - ▶ Intervenants: B. Bollig
 - ▶ Volume horaire: 12h

- ▶ Systèmes distribués temporisés
 - ▶ Intervenants: B. Bollig
 - ▶ Volume horaire: 6h

Reasoning about time: introductory remarks

Time and modalities

- ▶ Historical background: logical discussions of time and temporality goes back to ancient Greece.
- ▶ A modern approach using modalities (as in temporal logic) is due to A. Prior [Prior, 67].
- ▶ A **modality** is a syntactic object (a term) that modifies the relationships between a predicate and a subject.
- ▶ Examples of modalities:
 - ▶ It is necessary that P holds,
 - ▶ I know that P holds,
 - ▶ Tomorrow P holds,
 - ▶ It is an obligation that P holds,

Temporal logic in computer science

- ▶ Temporal logic in computer science has its beginning with A. Pnueli's seminal paper [Pnueli, 77] arguing that temporal logic can be used to reason about processes and computations.
- ▶ This is one of the most successful applications of logic to computer science.

Nature of time

- ▶ Fundamental question: ontological nature and structure of the flow of time.
- ▶ Numerous distinct alternatives exist (on-going debate since antiquity) and the answers to these alternatives are important for the development of the philosophy of nature.
- ▶ Different aspects of temporality in computer science lead to its own choices.
- ▶ Is time
 - ▶ linear or branching?
 - ▶ instant-based or interval-based?
 - ▶ discrete or continuous?
 - ▶ infinite? transfinite?
 - ▶ circular?
 - ▶ beginningless? endless? ...

Two traditional approaches

- ▶ In order to reason about time in a formal language we need to make assumptions about
 - ▶ the primitive time entities and,
 - ▶ the basic relations between them.
- ▶ Two traditional types of models of time: point-based and interval-based.
- ▶ The traditional logical language is first-order logic that can be viewed as the universal logic to be used in all fields of application.
- ▶ By contrast, temporal logics are adapted to models of computations and to specific applications.

Point-based approach

- ▶ Primitive temporal entities: instants and the basic relationships between them are equality and precedence.
- ▶ A flow of time is represented by $(T, <)$ (equality is implicit).
- ▶ First-order sentences expressing standard properties about the flow of time:

irreflexivity: $\forall x \neg(x < x)$,

transitivity: $\forall x, y, z (x < y \wedge y < z) \Rightarrow x < z$,

linearity: $\forall x, y (x = y \vee x < y \vee y < x)$,

no end: $\forall x \exists y (x < y)$,

density: between every two precedence-related instants there is another instant:

$\forall x, y (x < y \Rightarrow \exists z (x < z \wedge z < y))$.

First-order statements

- ▶ In order to talk about past and future events in first-order logic, we need a reference to “now”. To do so, let $N(x)$ mean “ x is the current instant”.
- ▶ Here are some statements:
 - ▶ “It has always been the case that ϕ ”:
$$N(x) \Rightarrow \forall y (y < x \Rightarrow \phi(y)).$$
 - ▶ “Sometime in the future it will be the case that ϕ ”:
$$N(x) \Rightarrow \exists y x < y \wedge \phi(y).$$
 - ▶ “ ϕ occurs infinitely often”
(some hypotheses on $(T, <)$ are required):
$$\forall x \exists y x < y \wedge \phi(y).$$

First-order logic is not enough

- ▶ First-order logic cannot express everything. For instance, it cannot express that a linear time flow is well-ordered.
- ▶ A totally ordered set (X, \leq) is well-ordered $\stackrel{\text{def}}{\iff}$ every non-empty set of X has a least element.
- ▶ $(\mathbb{N}, <)$ is well-ordered, $(\mathbb{Z}, <)$ is not.
- ▶ For this, one needs monadic second-order logic, where quantification over sets is allowed.
- ▶ Another standard property that is not first-order definable is to require that a relation R is the transitive closure of another relation R' .

Interval-based approach

- ▶ Primitive temporal entities: intervals and the basic relationships are equality, disjoint precedence, inclusion and overlap.

- ▶ A flow of time is represented as a structure of the form

$$(T, \prec, \sqsubseteq, O),$$

- ▶ First-order sentences expressing standard properties:
 - reflexivity of \sqsubseteq : $\forall x \ x \sqsubseteq x$,
 - anti-symmetry of \sqsubseteq : $\forall x, y \ (x \sqsubseteq y \wedge y \sqsubseteq x \Rightarrow x = y)$,
 - symmetry of O : $\forall x, y \ x O y \Rightarrow y O x$,
 - atomicity of \sqsubseteq : $\forall x \ \exists y \ (y \sqsubseteq x \wedge \forall z \ (z \sqsubseteq y \Rightarrow z = y))$.
- ▶ Proposals for interval temporal logics in AI (Allen's logic [Allen, 83]) and in applications to computer science such as real-time logics (see e.g. [Koymans, 90]).

Aspects of temporality in CS

Temporal phenomena in CS

Here are examples of the various ways in which reasoning about time and dealing with temporal phenomena is relevant and useful in computer science:

- ▶ specification and verification of concurrent and reactive systems,
- ▶ scheduling of the execution of programs by an operating system,
- ▶ synchronization of concurrent processes,
- ▶ temporal databases,
- ▶ real-time processes and systems.

Invariance, or safety properties

What must not happen throughout the computation will not happen.

- ▶ A train will not pass a red semaphore.
- ▶ The reactor will not overheat.
- ▶ Partial correctness properties: if a pre-condition P holds at the input of the program, then whenever it terminates, a post-condition Q will hold at the output.
- ▶ The counter x never takes the zero value.
- ▶ A nonce never takes twice the same value.

Eventuality, or liveness properties:

What has to happen during the computation will happen.

- ▶ Once a printing job is activated, eventually it will be completed.
- ▶ If a message is sent, eventually it will be delivered.
- ▶ Total correctness properties: if a pre-condition P holds at the input of the program, then it will terminate and a post-condition Q will hold at the output.

Fairness properties

- ▶ All processes will be treated fairly by the operating system (scheduler, ...).
- ▶ A typical situation: a process is enabled for the next step of its execution, and sends a request for scheduling. It may or may not be immediately scheduled for execution, because of competition with other processes.
- ▶ A fair scheduling would mean that if the process is persistent enough then eventually its request will be granted.

Transition systems

- ▶ Transition system $(S, (\xrightarrow{a})_{a \in Act})$:
 - ▶ S is a non-empty set of states,
 - ▶ Act is a non-empty set of actions,
 - ▶ \xrightarrow{a} is a binary relation in $S \times S$.
- ▶ Examples:
 - ▶ programs or processes run concurrently on the same computing device,
 - ▶ finite automata,
 - ▶ coffee machines,
 - ▶ Kripke frames (used in modal logic).

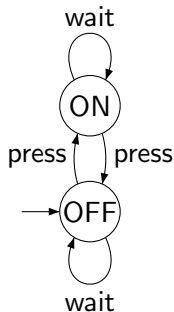
Labelled transition system

- ▶ A labelled transition system is a transition system augmented with a labelling: $S \rightarrow \mathcal{P}(\text{PROP})$ where PROP is a set of propositional variables (atomic properties).
- ▶ A run is a sequence of the form

$$s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \dots$$

- ▶ We may require that s_0 belongs to a set of initial states (for instance those in which all the counters are equal to zero).
- ▶ A run is also called a computation or an execution.

A simple LTS



▶ Executions:

- ▶ $\text{OFF} \xrightarrow{\text{wait}} \text{OFF} \xrightarrow{\text{wait}} \text{OFF} \xrightarrow{\text{wait}} \text{OFF} \xrightarrow{\text{wait}} \dots,$
- ▶ $\text{OFF} \xrightarrow{\text{wait}} \text{OFF} \xrightarrow{\text{press}} \text{ON} \xrightarrow{\text{press}} \text{OFF} \xrightarrow{\text{wait}} \dots,$
- ▶ $\text{OFF} \xrightarrow{\text{press}} \text{ON} \xrightarrow{\text{press}} \text{OFF} \xrightarrow{\text{press}} \text{ON} \xrightarrow{\text{press}} \dots,$

The language of temporal logic

Desirable properties of formal languages (I)

Temporal logic are formal specification languages with the following properties:

- ▶ to have an underlying flow of time in its models,
- ▶ to define mathematically the correctness of programs/systems,
- ▶ to express properties without ambiguity,
- ▶ to make formal proofs.

Desirable properties of formal languages (II)

Temporal logic well-suited for reactive systems:

- ▶ Temporal logic is specifically tailored for statements and reasoning which involve the notion of order in time.
- ▶ Compared with the mathematical formulas, temporal logic notation is clearer and simpler.
- ▶ Temporal logic states properties of executions from labelled transition systems.
- ▶ Reactive systems can be modeled by such transition systems especially since they do not necessarily compute values but maintain some interaction between different components.
- ▶ For instance, they include computer operating systems and network communication protocols.

Naive first-order approach

- ▶ First-order logic (with many sorts) can be easily used as a temporal logic assuming that the properties of the flow of time are definable in first-order logic.
- ▶ Atomic formula $P(t_1, \dots, t_n)$ where the t_i 's are terms and P is an n -ary predicate symbol, can be transformed into $P(t, t_1, \dots, t_n)$.
- ▶ The individual a always satisfies the property P can be written as $\forall t \in T P(t, a)$.
- ▶ First-order logic as a temporal logic suffers several drawbacks:
 - ▶ satisfiability for first-order logic is undecidable,
 - ▶ one may not need full expressive of the first-order logic but only a finite amount of temporal macros (“always”, “sometime”, “never”, ...),
 - ▶ second-order properties are also helpful.

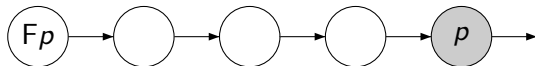
Temporal operators

- ▶ Temporal combinators allows one to speak about the sequencing of the states along an execution, rather than about the states taken individually.
- ▶ The simplest temporal combinators are X, F and G.
- ▶ We shall freely use the Boolean operators
 - ▶ \neg (negation),
 - ▶ \vee (disjunction),
 - ▶ \wedge (conjunction),
 - ▶ \Rightarrow (implication).
- ▶ Whereas ϕ states a property of the current state, $X\phi$ states that the next state (X for “next”) satisfies ϕ .



Sometime operator

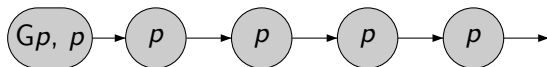
- ▶ Fp announces a future state (F for “future”) satisfies ϕ without specifying which state.
- ▶ This combinator can be read informally as “ ϕ will hold some day”.
- ▶ Foundations of the modal approach to temporal logic are set in [Prior, 67] in which are introduced the combinators F and G.



- ▶ First-order equivalence: $F\phi$ can be translated into $\exists y x \leq y \wedge \phi(y)$.

Always operator

- ▶ $G\phi$ states that all the future states satisfy ϕ .
- ▶ The operator G is the dual of F : whatever the formula ϕ may be, if ϕ is always satisfied, then it is not true that $\neg\phi$ will some day be satisfied.
- ▶ $G\phi$ and $\neg F\neg\phi$ are equivalent.



- ▶ If we (currently) are in a state of alert, then we will (later) be in a halt state:

$$G \text{ alert} \Rightarrow F \text{ halt}$$

Some other operators

- ▶ Likewise, the past operators F^{-1} (“sometime in the past”) and G^{-1} (“always in the past”) are also introduced in [Prior, 67].



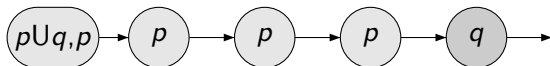
- ▶ Quantitative aspects:
 - ▶ Quantitative refinements for such operators are numerous.
 - ▶ For instance, $F_{]1,2[}\phi$ would mean that sometime in the future ϕ is satisfied within the time-interval $]1, 2[$.

Playing with temporal operators

- ▶ It is the ability to arbitrarily nest the various temporal combinators which gives temporal logic its power and its strength.
- ▶ Compare $G(\text{alert} \Rightarrow F \text{halt})$ with $G(\text{halt} \Rightarrow F^{-1} \text{alert})$.
- ▶ Nesting of F and G is very often used to express repetition properties.
- ▶ An ubiquitous property is to state that a property occurs infinitely often which can be written as: $GF\phi$.
- ▶ $F^\infty \phi \stackrel{\text{def}}{=} GF\phi$.
- ▶ The dual “almost always” temporal operator is defined by $G^\infty \phi \stackrel{\text{def}}{=} FG\phi$.

Until operator

- ▶ The U combinator (for “until”) is richer and more complicated than the combinator F.
- ▶ $\phi_1 U \phi_2$ states that ϕ_1 is verified until ϕ_2 is verified.



- ▶ $G(\text{alert} \Rightarrow F \text{halt})$ can be refined by

$$G(\text{alert} \Rightarrow (\text{alarm} U \text{halt})).$$

- ▶ Sometime operator: The F combinator is a special case of U: $F\phi$ and $\text{true}U\phi$ are equivalent.

Weak until

- ▶ “weak until” operator denoted W .
- ▶ The statement $\phi_1 W \phi_2$ still expresses “ $\phi_1 U \phi_2$ ”, but without the inexorable occurrence of ϕ_2 .
- ▶ If ϕ_2 never occurs, then ϕ_1 remains true until the end.
- ▶ $\phi_1 W \phi_2$ is equivalent to $G\phi_1 \vee (\phi_1 U \phi_2)$.

Other examples of expressions

(safety) $G\neg(x = 0)$,

(liveness) $G(p \Rightarrow Fq)$,

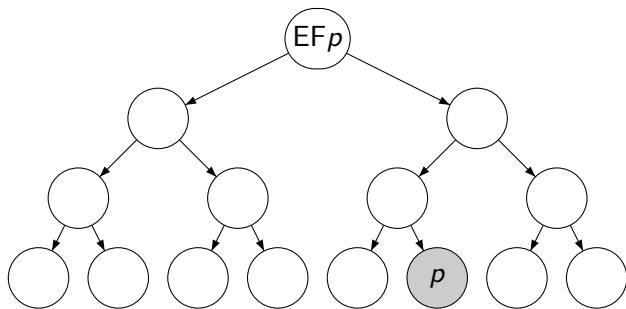
(total correctness) $(\text{init} \wedge p) \Rightarrow F(\text{end} \wedge q)$,

(strong fairness) $F^\infty \text{enabled} \Rightarrow F^\infty \text{executed}$.

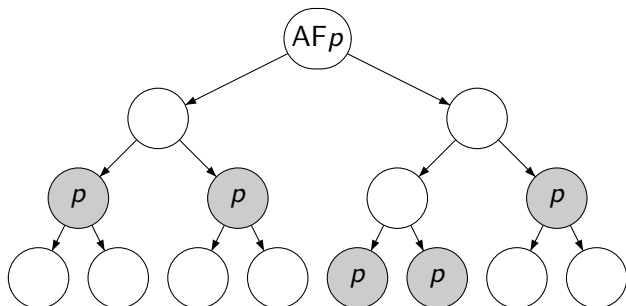
Path quantifiers

- ▶ Tree structure: many futures are possible starting from a given state.
- ▶ Special purpose quantifiers A and E allow one to quantify over the set of executions.
- ▶ Compare A with \forall in first-order logic.
- ▶ $A\phi$ states that all the executions out of the current state satisfy the property ϕ .
- ▶ Dually, $E\phi$ states that from the current state, there exists an execution satisfying ϕ .
- ▶ A and E quantify over paths whereas G and F quantify over positions a long a path.

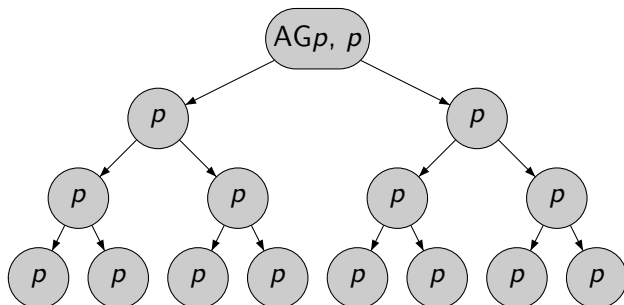
The expression $EF\phi$ states that it is possible (by following a suitable execution) to have ϕ some day.



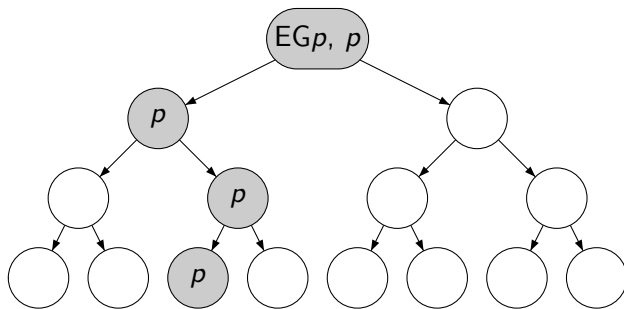
The expression $AF\phi$ states that we will necessarily have ϕ some day, regardless of the chosen execution.



The expression $AG\phi$ states that ϕ holds true in all future states including now.



The expression EG ϕ states that there is an execution on which ϕ holds always true.

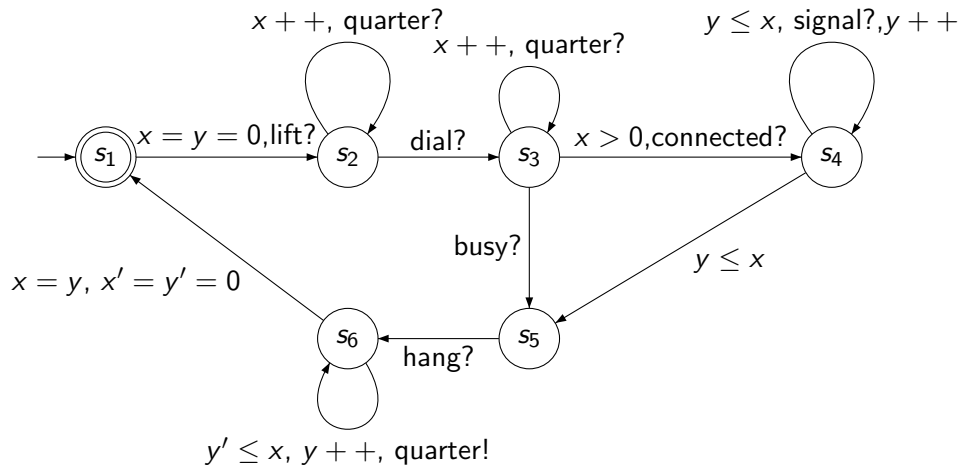


Branching-time vs. linear-time temporal logics

- ▶ “Branching-time logics” refers to logics which have this capability to freely quantify over the paths that are possible.
 - ▶ Computation Tree Logic CTL
 - ▶ CTL*
 - ▶ modal μ -calculus

- ▶ “Linear-time logics” refers to logics whose models are sequences/paths/executions.
 - ▶ Linear-time temporal logic LTL
 - ▶ LTL variants: ETL, linear μ -calculus, etc.

Example: pay phone controller [Comon & Cortier, CSL 00]



How to read the figure

- ▶ x is the number of quarters which have been inserted.
- ▶ y measures the total communication time.
- ▶ x' [resp. y'] is the new value of x [resp. y].
- ▶ The controller interacts with the environment. Messages followed by a question mark are received by the controller and messages followed by an exclamation mark are sent by the controller.

Finite representation of an infinite transition system

- ▶ A configuration of the controller is a triple (s, c_1, c_2) where s is a control state and c_1 [resp. c_2] is the value of x [resp. y].
- ▶ Because of the presence of messages, queues for messages should be added (omitted here).
- ▶ An execution is a (possibly infinite) sequence of configurations (constrained by the transitions of the controller).
- ▶ The automaton presented earlier is a finite and concise representation of an infinite labeled transition system (its interpretation).

Properties

- ▶ Communication time is never greater than the number of inserted quarters:

$$A G \neg(y > x).$$

- ▶ The number of quarters is infinitely often equal to zero:

$$A GF x = 0.$$

- ▶ There is an execution of the controller such that the communication time is always equal to zero:

$$E G y = 0.$$

- ▶ Whenever the communication is over, the controller regains the initial configuration:

$$A G (s_5 \Rightarrow F s_1).$$

- ▶ Whenever the control state s_1 is reached, $x = y = 0$ and conversely:

$$A G (s_1 \Leftrightarrow (x = 0 \wedge y = 0)).$$

Linear-time temporal logic LTL

Syntax

- ▶ LTL is the temporal logic of single computations, proposed by A. Pnueli in 1977.
- ▶ Formulae:

$$\phi, \psi ::= \overbrace{p_i \mid \neg\phi \mid \phi \wedge \psi \mid \phi \vee \psi}^{\text{propositional calculus}} \mid \overbrace{X\phi \mid F\phi \mid \phi U \psi}^{\text{temporal combinators}}$$

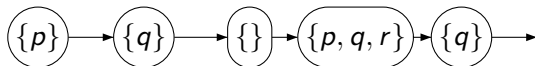
- ▶ $\text{PROP} = \{p_1, p_2, \dots\}$ is a countably infinite set of propositional variables.
- ▶ Elements of PROP are obtained by abstracting properties, for instance p may mean “ $x = 0$ ”.
- ▶ \vee and F are definable from other operators.
- ▶ $F\phi$ is equivalent to $\top U \phi$ (with $\top = p \vee \neg p$).

Other syntactic notions

- ▶ Abbreviations:
 - ▶ $G\phi$ for $\neg F\neg\phi$,
 - ▶ $\phi_1 \Rightarrow \phi_2$ for $\neg\phi_1 \vee \phi_2$,
 - ▶ $F^\infty\phi$ for $GF\phi$.
- ▶ Temporal depth of ϕ is the maximal number of temporal operators on a branch of the formula tree for ϕ .
- ▶ $LTL_n^k(H_1, H_2, \dots)$ denotes the LTL fragment restricted
 - ▶ to temporal operators in H_1, H_2, \dots ,
 - ▶ to formulae with temporal depth at most $k \geq 0$ and
 - ▶ with at most $n \geq 1$ propositional variables.
- ▶ Example: $LTL_\omega^2(F)$ is the set of LTL formulae with temporal depth at most 2 built from F .
- ▶ $|\phi|$ denotes the size of the formula ϕ viewed as a string of characters.

Semantics

- ▶ A structure (or model) for LTL is an infinite sequence $\sigma : \mathbb{N} \rightarrow \mathcal{P}(\text{PROP})$, i.e. an infinite word of $\mathcal{P}(\text{PROP})^\omega$.



- ▶ Satisfaction relation \models :
 - ▶ $\sigma, i \models p \stackrel{\text{def}}{\Leftrightarrow} p \in \sigma(i)$,
 - ▶ $\sigma, i \models \neg\phi \stackrel{\text{def}}{\Leftrightarrow} \sigma, i \not\models \phi$
(\vee and \wedge have their usual semantics),
 - ▶ $\sigma, i \models X\phi \stackrel{\text{def}}{\Leftrightarrow} \sigma, i+1 \models \phi$,
 - ▶ $\sigma, i \models F\phi \stackrel{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $\sigma, j \models \phi$,
 - ▶ $\sigma, i \models \phi U \psi \stackrel{\text{def}}{\Leftrightarrow}$ there is $j \geq i$ such that $\sigma, j \models \psi$ and for every $i \leq k < j$, we have $\sigma, k \models \phi$.

Satisfiability problem for LTL

- ▶ We write $\sigma \models \phi$ instead of $\sigma, 0 \models \phi$.
- ▶ $\text{Models}(\phi) \stackrel{\text{def}}{=} \{\sigma \in \mathcal{P}(\text{PROP})^\omega : \sigma \models \phi\}$.
- ▶ ϕ is LTL satisfiable $\stackrel{\text{def}}{\Leftrightarrow} \text{Models}(\phi) \neq \emptyset$.
- ▶ The satisfiability problem for LTL (denoted by $\text{SAT}(\text{LTL})$):
 - input:** an LTL formula ϕ ,
 - output:** 1 if there is σ such that $\sigma \models \phi$, 0 otherwise.
- ▶ In order to test satisfiability status of ϕ , restriction to propositional variables occurring in ϕ is sufficient.
 \Rightarrow Models of ϕ are ω -sequences over a finite alphabet.

Validity problem

- ▶ The validity problem for LTL:
 - input: an LTL formula ϕ ,
 - output: 1 if for every σ , $\sigma \models \phi$, 0 otherwise.
- ▶ ϕ is valid iff $\neg\phi$ is not satisfiable.
- ▶ Check validity for
 - ▶ $Fp \Leftrightarrow (p \vee XFp)$
 - ▶ $p_1Up_2 \Leftrightarrow (p_2 \vee (p_1 \wedge X(p_1Up_2)))$

Kripke structure

- ▶ Kripke structure $\mathcal{M} = (W, R, L)$
 - ▶ W is non-empty set of states,
 - ▶ R is a binary relation
(accessibility relation, one-step relation, transition relation),
 - ▶ L is a labeling $L : W \rightarrow \mathcal{P}(\text{PROP})$.
- ▶ \mathcal{M} is simply a directed graph for which each node is labeled by a propositional interpretation (labeled transition system).
- ▶ Path in \mathcal{M} : (finite or infinite) sequence $s_0s_1 \dots$ such that s_iRs_{i+1} for every $i \geq 0$.
- ▶ $\text{Paths}(\mathcal{M}, s_0)$: set of infinite paths of \mathcal{M} starting at the state s_0 .
- ▶ Alternatively, we also write $\text{Paths}(\mathcal{M}, s_0)$ to denote the set of infinite paths starting at s_0 on the alphabet $\mathcal{P}(\text{PROP})$. The first letter for such paths is therefore $L(s_0)$.

Model checking problem for LTL

- ▶ Size of (W, R, L) :

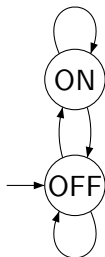
$$\text{card}(W) + \text{card}(R) + \sum_{w \in W} \text{card}(L(w)).$$

- ▶ Model checking problem for LTL, denoted by $\text{MC}^{\exists}(\text{LTL})$:
 - input:** an LTL formula ϕ , a finite and total Kripke structure \mathcal{M} and $s_0 \in W$,
 - output:** 1 if there is an infinite path starting at s_0 such that $\sigma \models \phi$ (written $\mathcal{M}, s_0 \models_{\exists} \phi$), 0 otherwise.
- ▶ Without any loss of generality, in the above statement we can assume that the codomain of the labeling L is restricted to the propositional variables occurring in ϕ .
- ▶ $\mathcal{M}, s_0 \models_{\exists} \phi$ iff $\text{Paths}(\mathcal{M}, s_0) \cap \text{Models}(\phi) \neq \emptyset$.

Universal model checking problem

- ▶ More useful problem for formal verification.
- ▶ Universal model checking problem for LTL, denoted by $MC^{\forall}(\text{LTL})$:
 - input:** an LTL formula ϕ , a finite and total Kripke structure \mathcal{M} and $s_0 \in W$,
 - output:** 1 if for every infinite path starting at s_0 , we have $\sigma \models \phi$ (written $\mathcal{M}, s_0 \models_{\forall} \phi$), 0 otherwise.
- ▶ By using standard properties about complexity classes about deterministic Turing machines one can show that $MC^{\forall}(\text{LTL})$ is in PSPACE [resp. PSPACE-hard] iff $MC^{\exists}(\text{LTL})$ is in PSPACE [resp. PSPACE-hard].

An elementary Kripke structure



Check which properties hold true:

- ▶ $\mathcal{M}, \text{ON} \models_{\exists} F^{\infty} \text{ON} \wedge F^{\infty} \text{OFF}$,
- ▶ $\mathcal{M}, \text{ON} \models_{\exists} \neg F^{\infty} \text{OFF}$,
- ▶ $\mathcal{M}, \text{ON} \models_{\exists} G(\text{ON} \Rightarrow \text{XX OFF})$.

Kamp's theorem on LTL

Strict until and strict since operators

- ▶ $\sigma, i \models \phi U^s \psi \stackrel{\text{def}}{\iff}$ there is $j > i$ such that $\sigma, j \models \psi$ and for every $i < k < j$, we have $\sigma, k \models \phi$.
- ▶ $\phi U \psi$ is equivalent to $\psi \vee (\phi \wedge (\phi U^s \psi))$.
- ▶ $\sigma, i \models \phi S^s \psi \stackrel{\text{def}}{\iff}$ there is $j < i$ such that $\sigma, j \models \psi$ and for every $j < k < i$, we have $\sigma, k \models \phi$.
- ▶ $X^{-1}p$ is equivalent to $\perp S^s p$.

First-order logic on $(\mathbb{N}, <)$

- ▶ Formulae:

$$\phi ::= p(x) \mid x < y \mid \neg\phi \mid \phi \wedge \psi \mid \exists x \phi$$

- ▶ First-order models: $(\mathbb{N}, <, p_1, p_2, \dots)$.
- ▶ These models can be viewed as LTL models.
- ▶ A valuation v maps variables to positions.
- ▶ Satisfaction relation:

$$(\mathbb{N}, <, p_1, p_2, \dots) \models_v \phi(x_1, \dots, x_k)$$

- ▶ $(\mathbb{N}, <, p_1, p_2, \dots) \models_v x < y \stackrel{\text{def}}{\iff} v(x) < v(y)$.

Expressive completeness

- ▶ For every LTL(U^s, S^s) formula ϕ , there is a first-order formula $\psi(x)$ such that for all σ and $i \in \mathbb{N}$,

$$\sigma, i \models \phi \text{ iff } \sigma \models_{[x \leftarrow i]} \psi(x)$$

- ▶ The (easy) proof is by defining a mapping that takes into account the first-order semantics of LTL(U^s, S^s).
- ▶ **Kamp's theorem:** For every first-order formula $\psi(x)$ with a unique free variable, there is an LTL(U^s, S^s) formula ϕ such that for all σ and $i \in \mathbb{N}$,

$$\sigma, i \models \phi \text{ iff } \sigma \models_{[x \leftarrow i]} \psi(x)$$

- ▶ The proof is quite complex [Kamp, PhD 68].

Dedekind-complete orders

- ▶ Dedekind-complete strict total order $(T, <)$: every non-empty bounded subset has a least upper bound.
- ▶ Examples:
 - ▶ $(\mathbb{N}, <)$, $(\mathbb{Z}, <)$, $(\mathbb{R}, <)$
 - ▶ ordinals (well-ordered sets)
- ▶ Counter-example: $(\mathbb{Q}, <)$
- ▶ Kamp's theorem is in fact established for every Dedekind-complete strict total order.

Complexity of LTL problems

Complexity classes

- ▶ $\text{co-PTIME} = \text{PTIME}$, $\text{PTIME} \subseteq \text{NP}$.
- ▶ It is open whether $\text{PTIME} \neq \text{NP}$.
- ▶ $\text{PSPACE} = \text{NPSPACE} = \text{co-PSPACE}$.
- ▶ $\text{PSPACE} \subseteq \text{EXPTIME}$, $\text{co-EXPTIME} = \text{EXPTIME}$ and $\text{PTIME} \neq \text{EXPTIME}$.
- ▶ It is open whether $\text{PSPACE} \neq \text{EXPTIME}$.
- ▶ See details in [Papadimitriou, Book 94].

Reductions

- ▶ Herein, hardness is defined wrt logarithmic space reductions.
- ▶ C -hardness of the problem \mathcal{P} is shown by reducing another problem already known to be C -hard.
- ▶ In order to show that a problem is in C , we can either design an ad hoc algorithm in C or reduce \mathcal{P} (in logarithmic space) to another problem already known to be in C (assuming that C contains LOGSPACE) .

From $MC^{\exists}(\text{LTL})$ to $\text{SAT}(\text{LTL})$

- ▶ There is a logarithmic space reduction from $MC^{\exists}(\text{LTL})$ to $\text{SAT}(\text{LTL})$.
- ▶ Idea of the proof: to encode a finite Kripke structure by an LTL formula [Sistla & Clarke, JACM 85].
- ▶ Input: finite total Kripke structure $\mathcal{M} = (W, R, L)$, $s_0 \in W$ and LTL formula ϕ built over p_1, \dots, p_k .
- ▶ To each state s of W , we associate a new propositional variable p_s and we encode the valuation $L(s)$ by the formula varprop_s below:

$$\text{varprop}_s \stackrel{\text{def}}{=} \bigwedge \{p_i : 1 \leq i \leq k, p_i \in L(s)\} \wedge \\ \bigwedge \{\neg p_i : 1 \leq i \leq k, p_i \notin L(s)\}.$$

Reduction (continued)

- ▶ For each state s , we encode the set of successors $R(s) = \{s' \in W : (s, s') \in R\}$ by the formula:

$$succ_s \stackrel{\text{def}}{=} X \bigvee \{p_{s'} : s' \in R(s)\}.$$

- ▶ Every state s is encoded by

$$\phi_s \stackrel{\text{def}}{=} p_s \Rightarrow (varprop_s \wedge succ_s)$$

- ▶ \mathcal{M} is encoded by

$$\phi_{\mathcal{M}} \stackrel{\text{def}}{=} G(\bigwedge \{\phi_s : s \in W\} \wedge \text{UNI})$$

- ▶ UNI states that a unique propositional variable from $\{p_s : s \in W\}$ is satisfied at the current state.
- ▶ $\mathcal{M}, s_0 \models \exists \phi$ iff $\phi_{\mathcal{M}} \wedge \phi \wedge p_{s_0}$ is satisfiable.

Exercise 1

1. $\phi[\psi]_\rho$ be an LTL formula with subformula ψ at the occurrence ρ . Show that $\phi[\psi]_\rho$ is satisfiable iff $\phi[p]_\rho \wedge G(p \Leftrightarrow \psi)$ is satisfiable where p is a new propositional variable no occurring in $\phi[\psi]_\rho$.
2. Conclude that there is a logarithmic space reduction from SAT(LTL) to SAT(LTL $^2_\omega$).
3. Show that SAT(LTL $_1(X)$) is NP-complete (independent of 1. and 2.).

Exercise II

- ▶ ϕ and ψ are initially equivalent ($\phi \equiv_0 \psi$) $\stackrel{\text{def}}{\Leftrightarrow}$ for every σ , $\sigma \models \phi$ iff $\sigma \models \psi$.
- ▶ ϕ and ψ are equivalent ($\phi \equiv \psi$) $\stackrel{\text{def}}{\Leftrightarrow}$ for all σ and $i \in \mathbb{N}$, $\sigma, i \models \phi$ iff $\sigma, i \models \psi$.
- ▶ For all LTL formulae ϕ , ψ , show that the propositions below are equivalent:
 1. $\phi \equiv_0 \psi$
 2. $\phi \equiv \psi$
 3. $\phi \Leftrightarrow \psi$ is valid
- ▶ What happens if we add past-time operators such as X^{-1} ?

Exercise III

Formalize the following specifications in LTL:

- ▶ The event ϕ will not occur before the event ψ which may or may not occur at all.
- ▶ The event ϕ will not occur before the event ψ which will occur.
- ▶ Every occurrence of the event ϕ is preceded by an occurrence of the event ψ , after the previous occurrence of ϕ , if any.
Question: can this specification be simplified to: "Between every two successive occurrences of the event ϕ there is an occurrence of the event ψ ."?