Logiques temporelles pour la vérification : expressivité, complexité, algorithmes

*Soutenance de Thèse – 03 avril 2003*

Nicolas MARKEY

Laboratoire d’Informatique Fondamentale d’Orléans
Formal verification?

- Why is verification crucial?
  - Reactive systems are everywhere,
  - They are ever more complex,
  - Numerous bugs have occurred (Ariane V, Therac-25, ...)

Formal verification?

• Why is verification crucial?
  – Reactive systems are everywhere,
  – They are ever more complex,
  – Numerous bugs have occurred (Ariane V, Therac-25, ...)

• Strict methods are necessary for verifying or certifying systems.
Formal verification?

• Why is verification crucial?
  – Reactive systems are everywhere,
  – They are ever more complex,
  – Numerous bugs have occurred (Ariane V, Therac-25, ...)

• Strict methods are necessary for verifying or certifying systems.

• Possible methods for formal verification:
  – Formal proof,
  – Testing,
  – Model checking...
Verification by model checking

System satisfies Property
Verification by model checking

System satisfies Property

Model Checker
Verification by model checking

System

? satisfies

Property

model of the System

Model Checker
Verification by model checking

- **System**: model of the System
- **Property**: Formula expressing the Property
- **Verification**: Does the System satisfy the Property?
Verification by model checking

System

? satisfies

Property

model of the System

Formula expressing the Property

Model Checker
Verification by model checking

System

? satisfies

Property

model of the System

Formula expressing the Property

Model Checker

yes / no
We use Kripke structures for modelling the system.
We use Kripke structures for modelling the system. For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.
For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
Kripke structures

We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.

For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system.
For instance, consider a (simplified) model of an A.T.M.:
We use Kripke structures for modelling the system. For instance, consider a (simplified) model of an A.T.M.:
Choosing a specification language

Pnueli: Using temporal logics for expressing properties.
Clarke, Sistla, Sifakis, Emerson: Model checking with temporal logics.
Lamport, Emerson: Many different temporal logics.
Choosing a specification language

Pnueli: Using temporal logics for expressing properties.
Clarke, Sistla, Sifakis, Emerson: Model checking with temporal logics.
Lamport, Emerson: Many different temporal logics.

Several criteria for comparing them:

- **Expressiveness**: Temporal logics have different expressive powers. This is an important criterion when choosing the temporal logic.
Choosing a specification language

Pnueli: Using temporal logics for expressing properties.
Clarke, Sistla, Sifakis, Emerson: Model checking with temporal logics.
Lamport, Emerson: Many different temporal logics.

Several criteria for comparing them:

- **Expressiveness**: Temporal logics have different expressive powers. This is an important criterion when choosing the temporal logic.

- **Succinctness**: Some properties can be expressed in several different temporal logics, but the formulas can be more or less long.
Choosing a specification language

Pnueli: Using temporal logics for expressing properties.
Clarke, Sistla, Sifakis, Emerson: Model checking with temporal logics.
Lamport, Emerson: Many different temporal logics.

Several criteria for comparing them:

- **Expressiveness**: Temporal logics have different expressive powers. This is an important criterion when choosing the temporal logic.

- **Succinctness**: Some properties can be expressed in several different temporal logics, but the formulas can be more or less long.

- **Complexity**: The problem of model checking a given temporal logic is more or less complex.
1. Past-time modalities in LTL
   We prove that past-time modalities do add succinctness to LTL, and that they really don’t change the complexity of model checking.

2. Extensions of CTL
   We give optimal algorithms for model checking $\text{CTL}^+$, $\text{FCTL}$, $\text{GFCTL}$ and $\text{ECTL}^+$: These problems are $\Delta^P_2$-complete.

3. Quantitative temporal logics
   We show that it is possible, in certain restricted cases, to perform timed model checking in polynomial time. We also study several other cases.

Conclusion
1. Past-time modalities in LTL
   We prove that past-time modalities do add succinctness to LTL, and that they really don't change the complexity of model checking.

2. Extensions of CTL
   We give optimal algorithms for model checking CTL\(^+\), FCTL, GFCTL and ECTL\(^+\): These problems are \(\Delta^P_2\)-complete.

3. Quantitative temporal logics
   We show that it is possible, in certain restricted cases, to perform timed model checking in polynomial time. We also study several other cases.

Conclusion
Definition of LTL +Past

LTL +Past (PLTL) is defined by the following syntax:

\[ \text{PLTL} \ni \varphi, \psi ::= \neg \varphi \mid \varphi \lor \psi \mid X \varphi \mid \varphi U \psi \mid X^{-1} \varphi \mid \varphi S \psi \mid p \mid q \mid \ldots \]

where \( p, q, \ldots \) are atomic propositions.
**Definition of LTL + Past**

**LTL + Past** (PLTL) is defined by the following syntax:

\[ \text{PLTL} \ni \varphi, \psi ::= \neg \varphi \mid \varphi \lor \psi \mid X \varphi \mid \varphi U \psi \mid X^{-1} \varphi \mid \varphi S \psi \mid p \mid q \mid \ldots \]

where \( p, q, \ldots \) are atomic propositions.

Some useful abbreviations:

\[ \begin{align*}
\top &\coloneqq p \lor \neg p \\
F \varphi &\coloneqq \top U \varphi \\
G \varphi &\coloneqq \neg F \neg \varphi \\
F^\infty \varphi &\coloneqq G F \varphi \\
G^\infty \varphi &\coloneqq F G \varphi \\
F^{-1} \varphi &\coloneqq \top S \varphi \\
G^{-1} \varphi &\coloneqq \neg F^{-1} \neg \varphi \\
I \varphi &\coloneqq G^{-1} F^{-1} \varphi
\end{align*} \]
Definition of LTL + Past

LTL + Past (PLTL) is defined by the following syntax:

\[
\text{PLTL} \ni \phi, \psi ::= \neg \phi \mid \phi \lor \psi \mid X \phi \mid \phi \cup \psi \mid X^{-1} \phi \mid \phi \iset \psi \mid p \mid q \mid \ldots
\]

where \( p, q, \ldots \) are atomic propositions.

Some useful abbreviations:

\[
\top \overset{\text{def}}{=} p \lor \neg p
\]

\[
F \phi \overset{\text{def}}{=} \top \cup \phi
\]

\[
G \phi \overset{\text{def}}{=} \neg F \neg \phi
\]

\[
\infty F \phi \overset{\text{def}}{=} G F \phi
\]

\[
G \phi \overset{\text{def}}{=} F G \phi
\]

Example:

\[
G (\text{give\_money} \Rightarrow F^{-1} \text{pin\_ok})
\]
Forgettable past [LS95]

\[ G (\text{give} \_\text{money} \Rightarrow F^{-1} \text{pin} \_\text{ok}) \]

This is not what we want to express: The following path satisfies the formula:

```
wait_card  ──→  pin_ok  ──→  give_money  ──→  wait_card  ──→  give_money
```

"\(F^{-1} \text{pin} \_\text{ok}\)" should only refer to what happened since the latest \(\text{wait} \_\text{card}\).
Forgettable past [LS95]

\[ \mathbf{G} (\text{give\_money} \Rightarrow \mathbf{F}^{-1} \text{pin\_ok}) \]

This is not what we want to express: The following path satisfies the formula:

```
wait\_card \rightarrow \text{pin\_ok} \rightarrow \text{give\_money} \rightarrow \text{wait\_card} \rightarrow \text{give\_money}
```

"\(\mathbf{F}^{-1} \text{pin\_ok}\)" should only refer to what happened since the latest \(\text{wait\_card}\).

Operator for "forgetting the past": \(\mathbf{N}\) (from now on):

\[ \pi, i \models \mathbf{N} \varphi \]
Forgettable past \cite{LS95}

\[ G (\text{give\_money} \Rightarrow F^{-1} \text{pin\_ok}) \]

This is not what we want to express: The following path satisfies the formula:

```
wait_card → pin_ok → give_money → wait_card → give_money
```

"\( F^{-1} \text{pin\_ok} \)" should only refer to what happened since the latest \text{wait\_card}.

Operator for "forgetting the past": \( N \) (from now on):

\[ \pi, i \models N \varphi \]

```
a → b → d → c → a → e
```

\[ N \varphi \]
Forgettable past [LS95]

\[ G(\text{give\_money} \Rightarrow \mathbf{F}^{-1} \text{pin\_ok}) \]

This is not what we want to express: The following path satisfies the formula:

```
| wait_card | pin_ok | give_money | wait_card | give_money |
```

"\( \mathbf{F}^{-1} \text{pin\_ok} \)" should only refer to what happened since the latest \text{wait\_card}.

Operator for "forgetting the past": \( \mathbf{N} \) (from now on):

\[ \pi, i \models \mathbf{N} \varphi \iff \pi \models^{i}, 0 \models \varphi \]
Forgettable past [LS95]

\[ G (\text{give\_money} \Rightarrow F^{-1} \text{pin\_ok}) \]

This is not what we want to express: The following path satisfies the formula:

- → \text{wait\_card} → \text{pin\_ok} → \text{give\_money} → \text{wait\_card} → \text{give\_money}

“\( F^{-1} \text{pin\_ok} \)” should only refer to what happened since the latest \( \text{wait\_card} \).

Operator for “forgetting the past”: \( N \) (from now on):

\[ \pi, i \models N \varphi \iff \pi^i, 0 \models \varphi \]

Example:

\[ G (\text{wait\_card} \Rightarrow N G (\text{give\_money} \Rightarrow F^{-1} \text{pin\_ok})) \]
Expressive power

\[ G(a \Rightarrow F^{-1} b) \equiv i \neg((\neg b) \cup (a \land \neg b)) \]
Expressive power

\[ G(a \Rightarrow F^{-1} b) \equiv_i \neg((\neg b) \mathbin{U} (a \land \neg b)) \]

Theorem [Kam68,GPSS80]: PLTL and LTL have the same expressive power.

Corollary [LMS02]: NLTL and LTL have the same expressive power.
Expressive power

\[ \mathbf{G} (a \Rightarrow F^{-1} b) \equiv_i \neg((\neg b) \mathbf{U} (a \land \neg b)) \]

Theorem [Kam68,GPSS80]: PLTL and LTL have the same expressive power.

Corollary [LMS02]: NLTL and LTL have the same expressive power.

[Gab87] provides an effective algorithm for translating a PLTL formula into an (initially) equivalent LTL formula.
Expressive power

\[ G(a \Rightarrow F^{-1}b) \equiv_i \neg((\neg b) \text{ U } (a \land \neg b)) \]

Theorem [Kam68,GPSS80]: PLTL and LTL have the same expressive power.

Corollary [LMS02]: NLTL and LTL have the same expressive power.

[Gab87] provides an effective algorithm for translating a PLTL formula into an (initially) equivalent LTL formula.

The LTL equivalent formula is much less intuitive. Moreover, the best known translation involves a triply-exponential increase in the size of the formula.
Theorem [Kam68,GPSS80]: PLTL and LTL have the same expressive power.

Corollary [LMS02]: NLTL and LTL have the same expressive power.

[Gab87] provides an effective algorithm for translating a PLTL formula into an (initially) equivalent LTL formula.

The LTL equivalent formula is much less intuitive. Moreover, the best known translation involves a triply-exponential increase in the size of the formula.

Can we avoid this explosion?
Succinctness of PLTL

Theorem [LMS02]: PLTL can be exponentially more succinct than LTL.
Theorem [LMS02]: PLTL can be exponentially more succinct than LTL.

Proof: Let \( \{p_0, p_1, \ldots, p_n\} \) be a set of atomic propositions.
Theorem [LMS02]: PLTL can be exponentially more succinct than LTL.

Proof: Let \( \{p_0, p_1, \ldots, p_n\} \) be a set of atomic propositions.

The PLTL formula

\[
\Phi \overset{\text{def}}{=} G \left[ \left( \bigwedge_{i=1}^{n} (p_i \leftrightarrow I p_i) \right) \Rightarrow (p_0 \leftrightarrow I p_0) \right]
\]

states that “any future state that agrees with the initial state on \( p_1, \ldots, p_n \) also agrees on \( p_0 \).”
Succinctness of PLTL

Theorem [LMS02]: PLTL can be exponentially more succinct than LTL.

Proof: Let \( \{p_0, p_1, \ldots, p_n\} \) be a set of atomic propositions.

The PLTL formula

\[
\Phi \overset{\text{def}}{=} G \left[ \left( \bigwedge_{i=1}^{n} (p_i \leftrightarrow I p_i) \right) \Rightarrow (p_0 \leftrightarrow I p_0) \right]
\]

states that “any future state that agrees with the initial state on \( p_1, \ldots, p_n \) also agrees on \( p_0 \)”.

Let \( \Psi \) be an LTL formula initially equivalent to \( \Phi \).

Therefore \( G \Psi \) expresses the following property:

“any two future states that agree on \( p_1, \ldots, p_n \) also agree on \( p_0 \)”
Succinctness of PLTL

Theorem [LMS02]: PLTL can be exponentially more succinct than LTL.

Proof: Let \( \{p_0, p_1, \ldots, p_n\} \) be a set of atomic propositions.

The PLTL formula

\[
\Phi \overset{\text{def}}{=} G \left[ \left( \bigwedge_{i=1}^{n} (p_i \iff I p_i) \right) \Rightarrow (p_0 \iff I p_0) \right]
\]

states that “any future state that agrees with the initial state on \( p_1, \ldots, p_n \) also agrees on \( p_0 \).”

Let \( \Psi \) be an LTL formula initially equivalent to \( \Phi \).

Therefore \( G \Psi \) expresses the following property:

“any two future states that agree on \( p_1, \ldots, p_n \) also agree on \( p_0 \)”

Any Büchi automaton recognizing that property has at least \( 2^{2^n} \) states. \[EVW97\]
The size of any LTL (or even PLTL) formula expressing that property is in \( \Omega(2^n) \).
Succinctness of NLTL

Theorem [LMS02]: NLTL can be exponentially more succinct than PLTL.
Succinctness of NLTL

**Theorem [LMS02]:** NLTL can be exponentially more succinct than PLTL.

**Proof:**

We still write

$$
\Phi \overset{\text{def}}{=} G \left[ \left( \bigwedge_{i=1}^{n} (p_i \Leftrightarrow I p_i) \right) \Rightarrow (p_0 \Leftrightarrow I p_0) \right]
$$

The NLTL formula $G N \Phi$ clearly states that "any two future states that agree on $p_1, \ldots, p_n$ also agree on $p_0$".

The size of any equivalent PLTL formula is in $\Omega(2^n)$. 
Model checking: Given $\varphi$ and a Kripke structure $K$, do we have, for any run $\pi$ of $K$: $\pi, 0 \models \varphi$?
Model checking fragments of NLTL

Model checking:
Given $\varphi$ and a Kripke structure $K$, do we have, for any run $\pi$ of $K$: $\pi, 0 \models \varphi$?

[SC85] proves that model checking is PSPACE-complete for LTL and PLTL.

Is past always for free?
Model checking fragments of NLTL

Model checking:

Given $\varphi$ and a Kripke structure $K$, do we have, for any run $\pi$ of $K$: $\pi, 0 \models \varphi$?

[SC85] proves that model checking is PSPACE-complete for LTL and PLTL.

Is past always for free?

szę complexity of fragments of NLTL.
Complexity of fragments of NLTL

NP-complete

PSPACE-complete

EXPSPACE-complete

$L(F)$

$L(F, F^{-1})$

$L(U)$

$L(U, S)$

$L(U, X)$

$L(F, X, F^{-1}, X^{-1})$

$L(F, X, F^{-1})$

$PLTL$

$L(F, F^{-1}, N)$

$LTL$

$NLTL$
Outline of the talk

1. Past-time modalities in LTL
   We prove that past-time modalities do add succinctness to LTL, and that they really don’t change the complexity of model checking.

2. Extensions of CTL
   We give optimal algorithms for model checking $\text{CTL}^+$, $\text{FCTL}$, $\text{GFCTL}$ and $\text{ECTL}^+$: These problems are $\Delta_2^P$-complete.

3. Quantitative temporal logics
   We show that it is possible, in certain restricted cases, to perform timed model checking in polynomial time. We also study several other cases.

Conclusion
Branching-time temporal logics

CTL [CE81,QS82]: Path quantification for all temporal modalities
Model checking is P-complete

Example: $\text{AG (EF } \text{card}_\text{back})$
CTL* [EH86]: Path quantification independent of temporal modalities
Model checking is PSPACE-complete [CES86]

Example: $\text{AF} (\text{ask}_\text{pin} \land \text{X} \text{check}_\text{pin})$
ECTL [EH86]: Allows $E \infty F$ and $A \infty F$
Strictly more expressive than CTL
Model checking is P-complete
Example: $E \infty F (\text{give\_money})$
CTL+ [EH85]: Boolean combinations in the scope of path quantifiers
Not more expressive than CTL, but exponentially more succinct [Wil99, AI01]
Model checking is NP-hard [CES86]
Example: \( E(G \neg \text{pin\_ok} \land F \text{give\_money}) \)
Branching-time temporal logics

FCTL $\rightarrow$ GFCTL

CTL $\rightarrow$ ECTL

CTL$^+$

CTL$^*$

FCTL, GFCTL [EL87]: Add fairness conditions to path quantifiers
Strictly more expressive than CTL
Model checking is NP-hard

Example: $A\overline{F_{\text{wrong\_pin}}} \land \overline{F_{\text{wait\_card}}}(F\text{\ give\_money})$
Branching-time temporal logics

ECTL+ [EH86]: Combines ECTL and CTL+ extensions
Strictly more expressive than ECTL and CTL+
Model checking is NP-hard
Example: $A(\infty F\text{wrong\_pin} \Rightarrow F\text{card\_back})$
Model checking $\text{CTL}^+$

- Model checking $\text{CTL}^+$ is NP-hard:

  $\text{SAT} : \text{is } (x \lor y \lor z) \land (\overline{x} \lor t \lor \overline{z}) \land (\overline{x} \lor \overline{t} \lor \overline{y})$ satisfiable ?

\[
\Phi = E((F \ x \lor F \ y \lor F \ z) \land (F \ \overline{x} \lor F \ t \lor F \ \overline{z}) \land (F \ \overline{x} \lor F \ \overline{t} \lor F \ \overline{y}))
\]
Model checking \( \text{CTL}^+ \)

- Model checking \( \text{CTL}^+ \) is \( \text{NP-hard} \):
  \[
  \text{SAT} : \text{is } (x \lor y \lor z) \land (\overline{x} \lor t \lor \overline{z}) \land (\overline{x} \lor \overline{t} \lor \overline{y}) \text{ satisfiable ?}
  \]
  \[
  \Phi = E((F x \lor F y \lor F z) \land (F \overline{x} \lor F t \lor F \overline{z}) \land (F \overline{x} \lor F \overline{t} \lor F \overline{y}))
  \]

- Model checking a formula \( E\varphi \in \text{CTL}^+ \), where \( \varphi \) has no path quantifier, can be done in \( \text{NP} \).

  Model checking \( \text{CTL}^+ \) is in \( \Delta_2^P = \text{P}^{\text{NP}} \).
Model checking \( \text{CTL}^+ \)

- Model checking \( \text{CTL}^+ \) is NP-hard:
  
  \[
  \text{SAT} : \text{is } (x \lor y \lor z) \land (\overline{x} \lor t \lor \overline{z}) \land (\overline{x} \lor \overline{t} \lor \overline{y}) \text{ satisfiable?}
  \]

  \[
  \Phi = E((F x \lor F y \lor F z) \land (F \overline{x} \lor F t \lor F \overline{z}) \land (F \overline{x} \lor F \overline{t} \lor F \overline{y}))
  \]

- Model checking a formula \( E\varphi \in \text{CTL}^+ \), where \( \varphi \) has no path quantifier, can be done in NP.

  Model checking \( \text{CTL}^+ \) is in \( \Delta^P_2 = \text{P}^{\text{NP}} \).

---

**Theorem [LMS01]**: Model checking \( \text{CTL}^+ \), \( \text{ECTL}^+ \) and \( \text{FCTL} \) is \( \Delta^P_2 \)-complete.
The SNSAT problem

Input:

\[ \mathcal{I} = \begin{cases} 
  x_1 := \exists Z_1 F_1(Z_1), \\
  x_2 := \exists Z_2 F_2(x_1, Z_2), \\
  \vdots \\
  x_n := \exists Z_n F_n(x_1, \ldots, x_{n-1}, Z_n) 
\end{cases} \]

\( \mathcal{I} \) defines a unique valuation \( v_\mathcal{I} \) of the variables in \( X \) where:

\[ v_\mathcal{I}(x_i) = \top \text{ iff } F_i(v_\mathcal{I}(x_1), \ldots, v_\mathcal{I}(x_{i-1}), Z_i) \text{ is satisfiable.} \]

Output: Does \( v_\mathcal{I}(x_n) = \top \)?
The SNSAT problem

Input:

\[ I = \begin{bmatrix}
    x_1 & := & \exists Z_1 F_1(Z_1), \\
    x_2 & := & \exists Z_2 F_2(x_1, Z_2), \\
    \vdots & : & \vdots \\
    x_n & := & \exists Z_n F_n(x_1, \ldots, x_{n-1}, Z_n)
\end{bmatrix} \]

\( I \) defines a unique valuation \( v_I \) of the variables in \( X \) where:

\[ v_I(x_i) = \top \iff F_i(v_I(x_1), \ldots, v_I(x_{i-1}), Z_i) \text{ is satisfiable.} \]

Output: Does \( v_I(x_n) = \top \)?

Theorem [LMS01]: SNSAT is \( \Delta^P_2 \)-complete.
Complexity of CTL⁺, ECTL⁺, FCTL

$\Delta^P_2$-hardness for CTL⁺:

Reduction from SNSAT: we build a Kripke structure in which a path represent a valuation of the variables, and a CTL⁺ formula expressing that a variable $x_i$ is true iff there is a witness that $F_i$ is satisfiable.
\( \Delta_2^P \)-hardness for CTL\(^+\):

Reduction from SNSAT: we build a Kripke structure in which a path represent a valuation of the variables, and a CTL\(^+\) formula expressing that a variable \( x_i \) is true iff there is a witness that \( F_i \) is satisfiable.
Outline of the talk

1. Past-time modalities in LTL
   We prove that past-time modalities do add succinctness to LTL, and that they really don’t change the complexity of model checking.

2. Extensions of CTL
   We give optimal algorithms for model checking $\text{CTL}^+$, $\text{FCTL}$, $\text{GFCTL}$ and $\text{ECTL}^+$: These problems are $\Delta^P_2$-complete.

3. Quantitative temporal logics
   We show that it is possible, in certain restricted cases, to perform timed model checking in polynomial time. We also study several other cases.

Conclusion
A **Durational Kripke Structure** is a Kripke structure whose transitions are labelled with a **duration**.
A Durational Kripke Structure is a Kripke structure whose transitions are labelled with a duration.
A Durational Kripke Structure is a Kripke structure whose transitions are labelled with a duration.

Example of timed temporal formulas:

\[ AG \text{ (} \text{pin\_ok} \Rightarrow EF_{\geq 8} \text{card\_back}) \]

\[ AG \ (EF_{\leq 60} \text{wait\_card}) \]
With or without exact durations

<table>
<thead>
<tr>
<th></th>
<th>ssDKS($\frac{0/1}{\rightarrow}$)</th>
<th>tight DKS, DKS</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>TCTL</strong></td>
<td>$\leq, \geq$</td>
<td>P-complete</td>
</tr>
<tr>
<td></td>
<td>$\leq, =, \geq$</td>
<td>P-complete</td>
</tr>
<tr>
<td></td>
<td></td>
<td>$\Delta^p_2$-complete</td>
</tr>
<tr>
<td><strong>TLTL</strong></td>
<td>$\leq, \geq$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td></td>
<td>$\leq, =, \geq$</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td><strong>TCTL</strong>*</td>
<td>$\leq, \geq$</td>
<td>PSPACE-complete</td>
</tr>
<tr>
<td></td>
<td>$\leq, =, \geq$</td>
<td>EXPSPACE-complete</td>
</tr>
<tr>
<td><strong>TCTL</strong></td>
<td>$\leq, \geq$</td>
<td>$\Delta^p_2$-complete</td>
</tr>
<tr>
<td></td>
<td>$\leq, =, \geq$</td>
<td></td>
</tr>
</tbody>
</table>
Outline of the talk

1. Past-time modalities in LTL
   We prove that past-time modalities do add succinctness to LTL, and that they really don’t change the complexity of model checking.

2. Extensions of CTL
   We give optimal algorithms for model checking $\text{CTL}^+$, $\text{FCTL}$, $\text{GFCTL}$ and $\text{ECTL}^+$: These problems are $\Delta_2^P$-complete.

3. Quantitative temporal logics
   We show that it is possible, in certain restricted cases, to perform timed model checking in polynomial time. We also study several other cases.

Conclusion
We solved several problems related to the expressiveness and complexity of various different logics.

- LTL should be extended with past modalities, since they make specification easier (more succinct and more natural), and are not harder to verify.

  The $N$ operator also brings succinctness, but verification becomes harder.
Conclusion

We solved several problems related to the expressiveness and complexity of various different logics.

- **LTL** should be extended with past modalities, since they make specification easier (more succinct and more natural), and are not harder to verify.

  The \( \mathbf{N} \) operator also brings succinctness, but verification becomes harder.

- In **CTL**, allowing the boolean combination of temporal statements (possibly fairness) in the scope of path quantifiers makes model checking much harder.

  These were the first verification problems known to be complete for \( \Delta^P_2 \).
Conclusion

We solved several problems related to the expressiveness and complexity of various different logics.

- **LTL** should be extended with past modalities, since they make specification easier (more succinct and more natural), and are not harder to verify.
  
The **N** operator also brings succinctness, but verification becomes harder.

- In **CTL**, allowing the boolean combination of temporal statements (possibly fairness) in the scope of path quantifiers makes model checking much harder.
  
These were the first verification problems known to be complete for $\Delta^P_2$.

- It is possible to perform timed model checking in **polynomial time**.
  
Model checking timed properties is harder when allowing exact constraints.
Future work

- still many open questions concerning expressiveness and complexity of temporal logics
- implementation of past modalities into LTL model checkers,
- model checking a single path,
- study different semantics for durations in DKS.