Theoretical Foundations of the UML
Lecture 16: A Logic for MSCs (Part 2)

Joost-Pieter Katoen

Lehrstuhl für Informatik 2
Software Modeling and Verification Group

moves.rwth-aachen.de/teaching/ss-20/fuml/

June 16, 2020
Outline

1. Introduction

2. Local Formulas and Path Expressions
   - Syntax
   - Formal Semantics

3. PDL Formulas

4. Verification problems for PDL
   - Model checking MSCs
   - Model checking CFMs
   - Model checking MSGs
   - Satisfiability

Propositional Dynamic Logic

\[
\text{MSC } M \quad \text{PDL-formula } \phi
\]

\[
M \models \phi?
\]

\[
\text{MSG } g \quad \text{PDL-formula } \phi
\]

\[
\forall M \in L(g), \quad M \models \phi?
\]

\[
\exists \text{MSC } M, \quad M \models \phi?
\]
Overview

1 Introduction

2 Local Formulas and Path Expressions
   • Syntax
   • Formal Semantics

3 PDL Formulas

4 Verification problems for PDL
   • Model checking MSCs
   • Model checking CFMs
   • Model checking MSGs
   • Satisfiability
Local formulas

**Definition (Syntax of local formulas)**

For communication action $\sigma \in Act$ and path expression $\alpha$, the grammar of **local formulas** is given by:

\[
\varphi ::= \text{true} \mid \sigma \mid \neg \varphi \mid \varphi \lor \varphi \mid \langle \alpha \rangle \varphi \mid \langle \alpha \rangle^{-1} \varphi
\]

The syntax of path expressions $\alpha$ will be defined later on.

**Definition (Derived operators)**

- $false := \neg true$
- $\varphi_1 \land \varphi_2 := \neg (\neg \varphi_1 \lor \neg \varphi_2)$
- $\varphi_1 \rightarrow \varphi_2 := \neg \varphi_1 \lor \varphi_2$
- $[\alpha] \varphi := \neg \langle \alpha \rangle \neg \varphi$
- $[\alpha]^{-1} \varphi := \neg \langle \alpha \rangle^{-1} \neg \varphi$

$\alpha$ is a regular expression that describes the possible admitted ways to navigate through a MSC.
Path expressions

Definition (Syntax of local formulas)

For communication action $\sigma \in Act$ and path expression $\alpha$, the grammar of local formulas is given by:

$$\varphi ::= true \mid \sigma \mid \neg \varphi \mid \varphi \lor \varphi \mid \langle \alpha \rangle \varphi \mid \langle \alpha \rangle^{-1} \varphi$$

Definition (Syntax of path expressions)

For local formula $\varphi$, the grammar of path expressions is given by:

$$\alpha ::= \{ \varphi \} \mid \text{proc} \mid \text{msg} \mid \alpha; \alpha \mid \alpha + \alpha \mid \alpha^*$$
PDL formulas

Definition (Syntax of PDL formulas)
For local formula $\varphi$, the grammar of PDL formulas is given by:

$$\Phi ::= \exists \varphi \mid \forall \varphi \mid \Phi \land \Phi \mid \Phi \lor \Phi$$

Negation

Negation is absent. As existential and universal quantification, as well as conjunction and disjunction are present, PDF-formulas are closed under negation.
Intuitive meaning of PDL formulas

- MSC $M$ satisfies $\exists \varphi$ if $M$ has some event $e$ satisfying $\varphi$

- MSC $M$ satisfies $\exists \langle \alpha \rangle \varphi$ if from some event $e$ in $M$, there exists an $\alpha$-labelled path from $e$ to an event $e'$, say, satisfying $\varphi$

- MSC $M$ satisfies $\exists [\alpha] \varphi$ if from some event $e$ in $M$, every event that can be reached via an $\alpha$-labelled path satisfies $\varphi$

\[ \equiv \exists e. \langle \alpha \rangle \varphi \]
Semantics of PDL formulas

Definition (Semantics of PDL formulas)
Let $M = (\mathcal{P}, E, C, l, m, <) \in \mathcal{M}$ be an MSC. 

$(M, \Phi) \in \models$ iff PDL formula $\Phi$ holds in MSC $M$.

- $M \models \exists \varphi$ iff $\exists e \in E. M, e \models \varphi$
- $M \models \forall \varphi$ iff $\forall e \in E. M, e \models \varphi$
- $M \models \Phi_1 \land \Phi_2$ iff $M \models \Phi_1$ and $M \models \Phi_2$
- $M \models \Phi_1 \lor \Phi_2$ iff $M \models \Phi_1$ or $M \models \Phi_2$
Example (1)

- The (unique) maximal event of $M$ is labeled by $?(2, 1, a)$  
  - Yes.  
  - No.
Example (1)

- The (unique) maximal event of $M$ is labeled by $?(2, 1, a)$  
  - Yes. No.

- $\forall (\langle (\text{proc} + \text{msg})^* \rangle ([\text{proc}] false \land ?(2, 1, a)))$  
  - Yes. No.
\[ M = \forall \langle (p+m)\rangle (\neg [p] \text{false} \land ?(2,1,a)) \]

iff (* semantics of PPL formulas *)

iff (* semantics of local formulas *)

\[ \forall e \in E. \left( \exists n \in \mathbb{N}. e \models \langle (p+m)\rangle (\neg [p] \text{false} \land ?(2,1,a)) \right) \]

intuitive: for every event in the MSG, there exists an event \( e' \) such that \( e \prec^* e' \) and

\[ e' \models [p] \text{false} \land ?(2,1,a) \]

\( e' \) has no successors at its process and it is labeled with ?(2,1,a)
Heft: OI holds → \( f \)\( \text{Lcptm} > \text{Ep} [f \text{false}] \)\( \text{falser} \)\( \text{fan}, \text{AD} \)\( I \)\( \text{take e}'s \)\( \text{eo} \)\( \text{eo} \)\( \text{K} \)\( \text{Ck} \)\( \text{sp} \)\( \text{sp} \)\( \text{eo} \)\( \text{en=z} \)\( G. \)\( \text{za} \)\( \text{and} \)\( \text{is} \)\( \text{the} \)\( \text{only} \) event in \( \text{Mcneff} \).

Thus, \( M \) left FOI false in \( G. \)\( \text{za} \)\( \text{and} \)\( \text{is} \)\( \text{the} \)\( \text{only} \) event in \( \text{Mcneff} \).

Thus \( M \) right \# OI.

\( \forall \theta \) holds \( A \)\( \text{Lcptm} > \text{Ep} \)\( \text{false} \)\( ? (\theta, \alpha) \).

\( \text{and} \)\( \text{similar} \)\( \text{for} \)\( \text{all} \)\( \text{other} \) events \( \text{in} \)\( \text{Mcneff} \).

Thus \( M \) right \# OI.

\( e' \models \)\( ? (\theta, \alpha) \).

\( \text{and} \)\( \text{similar} \)\( \text{for} \)\( \text{all} \)\( \text{other} \) events \( \text{in} \)\( \text{Mcneff} \).

Thus \( M \) right \# OI.
Example (2)

The maximal event on process 2 is labeled by ?(2, 1, a)  Yes. Yes.
\( M \models \exists E. (e \models [p] \text{false} \land ?(2,1,a)) \)

iff \( M \models \exists E. (e \models [p] \text{false} \land ?(2,1,a)) \)

iff \( M \models \exists E (e \models [p] \text{false} \land e \models ?(2,1,a)) \)

iff \( M \models \exists E (e \models [p] \text{false} \land e \models ?(2,1,a)) \)

iff \( M \models \exists E (\exists e' \in E. e \prec_p e' \land e' \models \text{false}) \)

and \( e(x) = ?(2,1,a) \)

iff \( M \models \exists E (\neg (\exists e' \in E. e \prec_p e') \land e(x) = ?(2,1,a)) \)

\( M \text{ left } \models \exists \) since \( e(x) = ?(2,1,a) \)

and \( e(x) \) has no successor at its process.

\( M \text{ right } \models \exists \) in a similar way using \( e(x) = e_0' \).
Example (2)

- The maximal event on process 2 is labeled by \( ?(2, 1, a) \)  
  Yes. Yes.

- \( \exists \left( [\text{proc}] \ false \land ?(2, 1, a) \right) \)  
  Yes. Yes.

Joost-Pieter Katoen
Theoretical Foundations of the UML
Example (3)

- No two consecutive events are labeled with $?(2, 3, c)$

- $\forall \left( \{ ?(2, 3, c) \}; \text{proc; \{} ?(2, 3, c) \}\right) \text{false}$
\[ M \models \forall [ \{ ?(2,3,c) \}, p \{ ?(2,3,c) \} ] \text{ false} \]

iff \[ \forall e \in E. \ e \not\models [ \text{ _ } ] \text{ false} \]

iff \[ (\text{ use that } [x] \varphi \equiv \neg \langle x \rangle \neg \varphi \upharpoonright ) \]

iff \[ \forall e \in E. \ e \not\models \neg \langle \text{ _ } \rangle \text{ false} \]

iff \[ \forall e \in E. \ \neg ( e \models \langle \text{ _ } \rangle \text{ true} ) \]

iff \[ \forall e \in E. \ \neg ( e \models \{ ?(2,3,c) \} \langle p \rangle \{ ?(2,3,c) \} \text{ true} ) \]

iff \[ \forall e \in E. \ \neg ( \ell(e) = ?(2,3,c) \land e \models \langle p \rangle \langle \text{ _ } \rangle \text{ true} ) \]

iff \[ \forall e \in E. \ \neg ( \ell(e) = ?(2,3,c) \text{ and } \exists e' \in E. \ e \prec_p e' \text{ and } \ell(e') = ?(2,3,c) ) \]

\[ M \leftarrow e \Rightarrow \text{ take } e = e_1 \text{ and } e' = e_2 \]

\[ e_1 \text{ and } e_2 \text{ violate the above formula} \]

\[ M \leftarrow \text{ two cases } e = e_1' \text{ and } e = e_2' \]

\[ e_1' \prec_p e_2' \text{ but } \ell(e_2') \neq ?(2,3,c) \]

\[ e_2' \prec_p e_0' \text{ but } \ell(e_0') \neq ?(2,3,c) \]
• The number of send events at process 3 is odd.
Abbreviations (auxiliary formulas)

!₁,ⱼ = ∨_{j ∈ C} !((₁, j, a))

(local formula)

message contents

p₁ = ∨_{j ∈ P, j ≠₁} (!₁,ⱼ ∨ ?₁,ⱼ)

actin at process 1

p₁ = p₁ iff e occurs at process 1
Path expression asserting that a certain event happens an even number of times

\( \Psi \) (local formula)

\[ \alpha = \left( \left( \{ \Psi \} \right) \left( \{ \Psi \} \right) \right)^* \]

1. No event satisfying \( \Psi \) occurs
2. No event satisfying \( \Psi \) occurs
3. No event satisfying \( \Psi \) occurs

\( \Psi \) occurs

\( \Psi \) occurs

\( \Psi \) occurs
• The number of send events at process 3 is odd.  
  No.  No.

• See next slide for a PDL-formula for a similar property.
MSC $M$ has an even number of messages sent from process 1 to 2:

$$\forall (\text{[proc]}^{-1} \text{false} \land P_1) \rightarrow \langle \alpha \rangle \quad \text{[proc] false} \land P_1$$

where $P_1 = \bigvee_{j \in P, j \neq 1} (!_{1,j} \lor ?_{1,j})$ with $!_{1,j} = \bigvee_{a \in C} !(1,j,a)$ and $?_{1,j}$ is defined in a similar way, i.e., $e \models P_1$ iff $e$ occurs at process 1.

Path expression $\alpha$ is defined by:

$$\alpha = ((\{\neg !_1\}; \text{proc})^{*}; \{!_1\}; \text{proc}; (\{\neg !_1\}; \text{proc})^{*}; \{!_1\}; \text{proc}; (\{\neg !_1\}; \text{proc})^{*})^{*}$$

and where $!_1$ abbreviates $\bigvee_{a \in C} !(1,2,a)$

$\neg = \text{no } !_1 \text{ event occurs}$
Let $i \neq j$

$$\forall \left( P_i \rightarrow ( < \text{proc}^*; \text{msg}; \text{proc}^*; \text{msg} > P_j ) \right)$$

expresses that process $i$ can "reach" process $j$ by exactly two messages (using intermediate processes).
Overview

1 Introduction

2 Local Formulas and Path Expressions
   - Syntax
   - Formal Semantics

3 PDL Formulas

4 Verification problems for PDL
   - Model checking MSCs
   - Model checking CFMs
   - Model checking MSGs
   - Satisfiability

given a \( \text{MSC} M \) + \emph{PDL-formula} \( \Phi \)

\textbf{Does} \( M \models \Phi \)?

\textbf{Does there exist a} \( \text{MSC} M \), \( M \models \Phi \)?
The following model-checking problem is **decidable** in polynomial time:

**Input:** MSC $M$, PDL-formula $\Phi$

**Output:** does $M \models \Phi$?

**Proof.**

(Sketch). Let $\Phi$ be a PDL formula. In subformulae $\langle \alpha \rangle \varphi$ and $\langle \alpha \rangle^{-1} \varphi$ of $\Phi$, view $\alpha$ as regular expression over finite alphabet \{proc, msg, $\{\varphi_1\}$, ..., $\{\varphi_n\}$\} with local formulae $\varphi_i$ (in $\Phi$). Any such expression can be transformed into a corresponding finite automaton of linear size. We proceed by inductively labelling events of the given MSC with states of the finite automata. This state information is then used to discover whether or not an event of $M$ satisfies a sub-formula $\langle \alpha \rangle \varphi$ and $\langle \alpha \rangle^{-1} \varphi$ which yields labellings in $\{0, 1\}$.  

[Joost-Pieter Katoen] Theoretical Foundations of the UML
The following model-checking problem is **decidable** in polynomial time:

**INPUT:** MSC $M$, PDL-formula $\Phi$

**OUTPUT:** does $M \models \Phi$?

**Proof.**

(Sketch). Let $\Phi$ be a PDL formula. In subformulae $\langle \alpha \rangle \varphi$ and $\langle \alpha \rangle^{-1} \varphi$ of $\Phi$, view $\alpha$ as regular expression over finite alphabet $\{\text{proc, msg, } \varphi_1, \ldots, \varphi_n\}$ with local formulae $\varphi_i$ (in $\Phi$). Any such expression can be transformed into a corresponding finite automaton of linear size. We proceed by inductively labelling events of the given MSC with states of the finite automata. This state information is then used to discover whether or not an event of $M$ satisfies a sub-formula $\langle \alpha \rangle \varphi$ and $\langle \alpha \rangle^{-1} \varphi$ which yields labellings in $\{0, 1\}$. Boolean combinations and $\exists \varphi$ and $\forall \varphi$ are then handled in a straightforward manner. Time complexity: $O(|E| \cdot |\Phi|^2)$ with $|E|$ is the number of events in $M$ and $|\Phi|$ the length of $\Phi$. 

[Joost-Pieter Katoen] Theoretical Foundations of the UML
PDL model checking algorithm for MSCs (1)

```
LOCAL FORMULA CHECK:
1  V = {0, .., n-1}
2
3  boolean[] Sat(LocalFormula f) {
4    boolean[] sat = new boolean[n];
5    switch(f) {
6      case Not(f1):
7        boolean[] sat1 = Sat(f1);
8        for (int i = 0; i < n; i++)
9          sat[i] = !sat1[i];
10       break;
11      case Or(f1, f2):
12        boolean[] sat1 = Sat(f1);
13        boolean[] sat2 = Sat(f2);
14        for (int i = 0; i < n; i++)
15          sat[i] = sat1[i] || sat2[i];
16        break;
17      case Event(..):
18        for (int i = 0; i < n; i++)
19          sat[i] = (V[i].event.equals(f));
20       break;
```
PDL model checking algorithm for MSCs (2)

21 case <p1> f2:
22     boolean[][] trans1 = Trans(p1);
23     boolean[] sat2 = Sat(f2);
24     for (int i = 0; i < n; i++) {
25         sat[i] = false;
26         for (int j = 0; j < n; j++)
27             if(trans[i][j])
28                 sat[i] = sat2[j];
29     }
30     break;
31 case <p1>⁻¹ f2:
32     boolean[][] trans1 = TransBack(p1);
33     boolean[] sat2 = Sat(f2);
34     for (int i = 0; i < n; i++) {
35         sat[i] = false;
36         for (int j = 0; j < n; j++)
37             if(trans[i][j])
38                 sat[i] = sat2[j];
39     }
40     break;
41 }
42 }
PDL model checking algorithm for MSCs (3)

trans[i][j] = true
iff (e_i, e_j) \models p

(\text{e, } e') \models p

// concatenation

// choice

\text{FORWARD PATH EXPRESSION CHECK:}

1. boolean[][] Trans(PathFormula p) {
2. boolean[][] trans = new boolean[n][n];
3. switch(p) {
4. case (p1; p2):
5. boolean[][] trans1 = Trans(p1);
6. boolean[][] trans2 = Trans(p2);
7. for (int i = 0; i < n; i++)
8. for (int k = 0; k < n; k++) {
9. trans[i][k] = false;
10. for (int j = 0; j < n; j++)
11. if(trans1[i][j] && trans1[i][k])
12. trans[i][k] = true;
13. }
14. break;
15. case p1 + p2:
16. boolean[][] trans1 = Trans(p1);
17. boolean[][] trans2 = Trans(p2);
18. for (int i = 0; i < n; i++)
19. for (int j = 0; j < n; j++)
20. trans[i][j] = trans1[i][j] || trans2[i][j];
21. break;
case p1*:  
  boolean[][] trans1 = Trans(p1);
  for (int i = 0; i < n; i++)
    for (int j = 0; j < n; j++)
      star[i][j] = (i==j);
  while (true) {
    for (int i = 0; i < n; i++)
      for (int j = 0; j < n; j++)
        if (trans1[i][j])
          for (int k = 0; k < n; k++)
            if (!trans[i][k] && trans1[j][k]) {
              trans[i][k] = true;
              continue;
            }
        break;
  }
}
Communication finite-state machines

Let a CFM now be accepting if all its processes have reached a local accepting state and either halt there or visit a local accepting state infinitely often.

An example CFM and an infinite MSC accepted by it

Client-server interaction to get access to an interface. Accepting state is \((s_3, t_0, q_0)\).
A CFM is accepting if all its processes have reached a local accepting state and reside their ad infinitum.

The language $\mathcal{L}(A)$ of CFM $A$ is the set of MSCs that admit an accepting run.

**CFM versus PDL**

A CFM $A$ satisfies PDL-formula $\Phi$, denoted $A \models \Phi$, whenever for all MSCs $M$ it holds: $M \in \mathcal{L}(A)$ if and only if $M \models \Phi$.

The example CFM satisfies $\forall (P_1 \rightarrow (\langle \text{proc}^*; \text{msg}; \text{proc}^*; \text{msg} \rangle P_3)$ where for $i \in \mathcal{P}$, formula $P_i = \bigvee_{j \in \mathcal{P}, j \neq i} (i, j \lor i, j)$, i.e., $M, e \models P_i$ iff $e$ occurs at process $i$. The PDL formula asserts that process 3 (Interface) can be “reached” from 1 (Client) by exactly two messages using an intermediate process in between.
Model checking CFMs versus PDL

The following model-checking problem is undecidable:

**INPUT:** a CFM $\mathcal{A}$, PDL-formula $\Phi$

**OUTPUT:** is there an MSC $M \in L(\mathcal{A})$ with $M \models \Phi$?

**Proof.**

Follows immediately from the fact that the emptiness problem for CFMs is undecidable. By using the formula $true$, the above problem encodes the emptiness problem.

$$\forall true \\
\exists true$$
The following model-checking problem is **undecidable**:

**INPUT:** a CFM $A$, PDL-formula $\Phi$

**OUTPUT:** is there an MSC $M \in \mathcal{L}(A)$ with $M \models \Phi$?

**Proof.**

Follows immediately from the fact that the emptiness problem for CFMs is undecidable. By using the formula $true$, the above problem encodes the emptiness problem.

To obtain decidable model-checking problems, we consider **$B$-bounded** MSCs.
The following model-checking problem is PSPACE-complete:

**INPUT:** a CFM $A$ and $B \in \mathbb{N}_{>0}$, PDL-formula $\Phi$

**OUTPUT:** is there an $\exists B$-bounded MSC $M \in \mathcal{L}(A)$ with $M \models \Phi$?

**Proof.**

(Sketch). Every PDL formula $\Phi$ can effectively be translated into a CFM $A_\Phi$ such that $A_\Phi \models \Phi$. Construction is involved.

$$\forall \Phi. M = \{ M \in M \mid M \models \Phi \}$$ can be accepted by a CFM $A$ such that $L(A) = M$. 
The following model-checking problem is PSPACE-complete:

**Input:** a CFM $A$ and $B \in \mathbb{N}_{>0}$, PDL-formula $\Phi$

**Output:** is there an $\exists B$-bounded MSC $M \in \mathcal{L}(A)$ with $M \models \Phi$?

**Proof.**

(Sketch). Every PDL formula $\Phi$ can effectively be translated into a CFM $A_\Phi$ such that $A_\Phi \models \Phi$. The details are out of the scope of this lecture. This synthesis step is independent of the channel bound size $B$ (if any). The size of $A_\Phi$ is exponential in the length of $\Phi$ and the number of processes in $P$.

$\Phi \xrightarrow{\text{CFM}} A_\Phi \quad |A_\Phi| \in O(2^{14})$
The following model-checking problem is PSPACE-complete:

**Input:** a CFM $\mathcal{A}$ and $B \in \mathbb{N}_{>0}$, PDL-formula $\Phi$

**Output:** is there an $\exists B$-bounded MSC $M \in \mathcal{L}(\mathcal{A})$ with $M \models \Phi$?

**Proof.**

(Sketch). Every PDL formula $\Phi$ can effectively be translated into a CFM $\mathcal{A}_\Phi$ such that $\mathcal{A}_\Phi \models \Phi$. The details are out of the scope of this lecture. This synthesis step is independent of the channel bound size $B$ (if any). The size of $\mathcal{A}_\Phi$ is exponential in the length of $\Phi$ and the number of processes in $\mathcal{P}$. Then construct a CFM accepting $\mathcal{L}(\mathcal{A}) \cap \mathcal{L}(\mathcal{A}_\Phi)$.
The following model-checking problem is PSPACE-complete:

**INPUT:** a CFM $A$ and $B \in \mathbb{N}_{\geq 0}$, PDL-formula $\Phi$

**OUTPUT:** is there an $\exists B$-bounded MSC $M \in \mathcal{L}(A)$ with $M \models \Phi$?

**Proof.**

(Sketch). Every PDL formula $\Phi$ can effectively be translated into a CFM $A_\Phi$ such that $A_\Phi \models \Phi$. The details are out of the scope of this lecture. This synthesis step is independent of the channel bound size $B$ (if any). The size of $A_\Phi$ is exponential in the length of $\Phi$ and the number of processes in $P$. Then construct a CFM accepting $\mathcal{L}(A) \cap \mathcal{L}(A_\Phi)$. Decide whether the resulting CFM accepts some $\exists B$-bounded MSC. This can all be done in polynomial space. The PSPACE-hardness follows from the hardness of LTL model checking.
The following model-checking problem is PSPACE-complete:

**INPUT:** a MSG $G$ and PDL-formula $\Phi$

**OUTPUT:** is there an MSC $M \in \mathcal{L}(G)$ with $M \models \Phi$?

**Proof.**

(Sketch.) For every vertex $v$, we can determine a linearization of the MSC $\lambda(v)$.

Construct a finite automaton $A_G$ that accepts a linearization for every $M \in \mathcal{L}(G)$, and vice versa, each word accepted by $A_G$ is a linearization of some $M \in \mathcal{L}(G)$. The size of $A_G$ is linear in the size of $G$. Construct a CFM $A_\Phi$ for PDL-formula $\Phi$ with $M \in \mathcal{L}(A_\Phi)$ iff $M \models \Phi$. Construct a transition system by running $A_G$ and $A_\Phi$ simultaneously. This construction terminates as $A_G$ only accepts linearizations that are $B$-bounded (as every linearization of MSG $G$ is $\exists B$-bounded by definition).

Deciding whether some simultaneous run is accepting can be done in polynomial space. The PSPACE-hardness follows from the hardness of LTL model checking.
Satisfiability problem for MSCs

Model checking MSCs versus PDL

[Kern, 2009]

The following model-checking problem is **decidable** in polynomial time:

**Input:** MSC $M$, PDL-formula $\Phi$

**Output:** does $M \models \Phi$?

MSC satisfiability for PDL

[Bollig et. al, 2011]

The following satisfiability problem is **undecidable**:

**Input:** PDL-formula $\Phi$

**Output:** is there an MSC $M$ with $M \models \Phi$?
Other PDL decision problems

Theorem: [Alur et al., 2001, Bollig et al., 2007]

Let \( \Phi \) be a PDL formula. Then:

1. The decision problem “does there exist a CFM \( \mathcal{A} \) such that for any MSC \( M \in \mathcal{L}(\mathcal{A}) \) we have \( M \models \Phi \)” is undecidable.

2. The decision problem “does there exist a CFM \( \mathcal{A} \) such that for some \( \exists B \)-bounded MSC \( M \in \mathcal{L}(\mathcal{A}) \) we have \( M \models \Phi \)” is decidable in PSPACE.

3. The decision problem “for MSG \( G \), is there an MSC \( M \in \mathcal{L}(G) \) such that \( M \models \Phi \)” is NP-complete.
## PDL Verification Problems

<table>
<thead>
<tr>
<th>MSC $M$</th>
<th>PDL-formula $\Phi$</th>
<th>$M \models \Phi$?</th>
<th>Decidable in P</th>
</tr>
</thead>
</table>

<table>
<thead>
<tr>
<th>CFM $A$</th>
<th>PDL-formula $\Phi$</th>
<th>$\exists M \in L(A). M \models \Phi$?</th>
<th>Undecidable</th>
</tr>
</thead>
</table>

| CFM $A$ | PDL-formula $\Phi$ | $\exists M \in L(A). M \models \Phi$ and $M$ is $\exists B$-bounded? | Decidable (PSPACE-complete) |

| MSG $G$ | PDL-formula $\Phi$ | $\exists M \in L(G). M \models \Phi$? | Decidable (PSPACE-complete) |

## PDL Satisfiability Problems

<table>
<thead>
<tr>
<th>PDL-formula $\Phi$</th>
<th>$\exists M \in L(A). M \models \Phi$?</th>
<th>Undecidable</th>
</tr>
</thead>
</table>

| PDL-formula $\Phi$ | $\exists \text{CFM } A$ such that $\forall M \in L(A). M \models \Phi$? | Undecidable |

| PDL-formula $\Phi$ | $\exists \text{CFM } A$ such that bound $B \in \mathbb{N}$ $\exists M \in L(A). M$ is $\exists B$-bounded $M \models \Phi$? | Decidable in PSPACE |
For logic-interested people:

1. \[ \text{PDL} \not\subseteq \text{MSO-logic} \]
   - monadic second order logic
   - \( \exists x \forall y \)
   - \( \exists X \forall Y \exists x \in X \).

2. Extending PDL with intersection yields a logic that is more expressive than CFMs.

\[ \langle x_1 \cup x_2 \rangle \mathcal{P} \]

"there exist two paths described by \( x_1 \) and \( x_2 \) resp. that both lead to an event satisfying \( \mathcal{P} \)"

Then [Bollig et al., 2010] showed:

\[ \exists \mathcal{P} \text{ such that } \{ M \in M \mid M \models \mathcal{P} \} \]

in extended PDL cannot be accepted by a CFM."
③ PDL supports "forward" navigation $<\alpha>\varphi$

and "backward" navigation $<\alpha>\neg\varphi$. PDL does not allow to mix "forward" and "backward" in a single formula

e.g. $\text{proc}^*; \text{msg}^{-1}; \text{proc}$ is not a syntactically admitted formula.

④ The temporal logic formula $\psi$ until $\phi$, i.e. $\psi$ holds at all events until an event satisfying $\phi$ is "reached" can be expressed as PDL-formula

$$<\text{proc}^*; (\text{proc} + \text{msg})^* > \psi$$