On Planners as Beyond NP Solvers

Malte Helmert

University of Basel

AAAI 2016 Workshop on Beyond NP
POP QUIZ: Automata Theory
Example

Given: a DFA $M$

Question: Is $L(M)$ empty?

How can we decide this? How difficult is it?
Example

Given: two DFAs $M_1$ and $M_2$
Question: Is $L(M_1) \cap L(M_2)$ empty?

How can we decide this? How difficult is it?
DFA Pop Quiz: \( n \)

Given: DFAs \( M_1, \ldots, M_n \)

Question: Is \( \bigcap_{i=1}^{n} L(M_i) \) empty?

How can we decide this? How difficult is it?
Empty intersection for $n$ DFAs is PSPACE-complete (Kozen 1977).

It is a trivial syntactic variant of the plan existence problem for domain-independent classical planning.

We will get to planning shortly.
About This Talk
Why I am speaking here today:
History of This Talk

Why I am speaking here today:

- **Adnan**: “Can you forward the Beyond NP workshop call to the planning community?”
History of This Talk

Why I am speaking here today:

- **Adnan**: “Can you forward the Beyond NP workshop call to the planning community?”
- **Me**: “They won’t like this part:”

From the CfP

“QBF solvers [...] are now established as the prototypical solvers for the complexity class PSPACE.”
History of This Talk

Why I am speaking here today:

- **Adnan**: “Can you forward the Beyond NP workshop call to the planning community?”
- **Me**: “They won’t like this part:”

From the CfP

“QBF solvers [...] are now established as the prototypical solvers for the complexity class PSPACE.”

- **Adnan**: “Why not?”
Why I am speaking here today:

- **Adnan**: “Can you forward the Beyond NP workshop call to the planning community?”
- **Me**: “They won’t like this part:”

From the CfP

“QBF solvers [...] are now established as the prototypical solvers for the complexity class PSPACE.”

- **Adnan**: “Why not?”
- **Me**: “Planning is also a pure and abstract problem and a popular compilation target for PSPACE-complete problems!”
History of This Talk

Why I am speaking here today:

- **Adnan**: “Can you forward the Beyond NP workshop call to the planning community?”
- **Me**: “They won’t like this part:”

From the CfP

“QBF solvers [...] are now established as the prototypical solvers for the complexity class PSPACE.”

- **Adnan**: “Why not?”
- **Me**: “Planning is also a pure and abstract problem and a popular compilation target for PSPACE-complete problems!”
- **Adnan**: “Go tell us about it!”
Guiding Questions

This talk:

- What is planning?
- Why is it a general problem?
- How does it fit into the Beyond NP landscape?
What is Planning?
this talk: “planning” = domain-independent classical planning
(Domain-Independent Classical) Planning

this talk: “planning” = domain-independent classical planning

Planning (pithy definition)

“Planning is the art and practice of thinking before acting.”

— Patrik Haslum
(Domain-Independent Classical) Planning

this talk: “planning” = domain-independent classical planning

Planning (pithy definition)

“Planning is the art and practice of thinking before acting.”
— Patrik Haslum

Planning (more technical definition)

“Selecting a goal-leading course of action based on a high-level description of the world.”
— Jörg Hoffmann
this talk: “planning” = domain-independent classical planning

Planning (pithy definition)
“Planning is the art and practice of thinking before acting.”
— Patrik Haslum

Planning (more technical definition)
“Selecting a goal-leading course of action based on a high-level description of the world.”
— Jörg Hoffmann

Domain-Independence of Classical Planning
One planning algorithm for all relevant domains.
Example: The Seven Bridges of Königsberg

image credits: Bogdan Giușcă (public domain)
Example: Deep Space One

image credits: NASA (public domain)
Example: Intelligent Greenhouse
Example: FreeCell

image credits: GNOME Project (GNU General Public License)
Classical Planning as Reachability in Transition Systems

classical planning:
can be seen as finding paths in implicitly defined digraphs
Classical Planning as Reachability in Transition Systems

classical planning:
can be seen as finding paths in implicitly defined digraphs
classical planning:
can be seen as finding paths in large implicitly defined digraphs

Example problem sizes:
- elevator control: $6.92 \cdot 10^{19}$ reachable states
- greenhouse automation: $1.68 \cdot 10^{21}$ reachable states
- transportation logistics: $6.31 \cdot 10^{218}$ reachable states
Planning Task in a Typical Representation

- finite-domain state variables:
  \[ x \in \{a, b, c\}, \ y \in \{a, b\}, \ z \in \{a, b, c\} \]
Planning Task in a Typical Representation

- **finite-domain state variables:**
  
  \( x \in \{a, b, c\}, \ y \in \{a, b\}, \ z \in \{a, b, c\} \)

- **initial state:**
  
  \( \{x \mapsto a, \ y \mapsto a, \ z \mapsto a\} \)
Planning Task in a Typical Representation

- finite-domain state variables:
  \[ x \in \{a, b, c\}, \ y \in \{a, b\}, \ z \in \{a, b, c\} \]

- initial state:
  \[ \{x \mapsto a, \ y \mapsto a, \ z \mapsto a\} \]

- goal:
  \[ \{x \mapsto c, \ z \mapsto b\} \]
Planning Task in a Typical Representation

- finite-domain state variables:
  \( x \in \{a, b, c\} \), \( y \in \{a, b\} \), \( z \in \{a, b, c\} \)

- initial state:
  \( \{x \mapsto a, y \mapsto a, z \mapsto a\} \)

- goal:
  \( \{x \mapsto c, z \mapsto b\} \)

- actions:
  \( a_1 : x = a, y = a \rightarrow y := b, z := c \)
  \( a_2 : \ldots \)
  \( \ldots \)
Planning Task in a Typical Representation

- finite-domain state variables:
  $x \in \{a, b, c\}$, $y \in \{a, b\}$, $z \in \{a, b, c\}$

- initial state:
  \{x \mapsto a, y \mapsto a, z \mapsto a\}

- goal:
  \{x \mapsto c, z \mapsto b\}

- actions:
  \begin{align*}
  a_1 & : \ x = a, \ y = a \quad \rightarrow \quad \ y := b, \ z := c \\
  a_2 & : \quad \ldots \\
  \ldots & \\
  \end{align*}

Objective:

- find sequence of actions transforming initial state to state consistent with the goal
- optionally: minimize length/cost of path
Why is Planning a General Problem?
next slides: five reasons why planning is a suitable representative problem in the spirit of Beyond NP

for simplicity: focus on plan existence decision problem: “Is a given instance solvable?”
Reason 1: Planning is PSPACE-Complete

Planning is PSPACE-complete.

OK, that’s a start. But that’s not really enough, is it?
Reason 1: Planning is PSPACE-Complete

Planning is PSPACE-complete.

OK, that’s a start. But that’s not really enough, is it?

What makes a good Beyond NP problem?

- should be clean and simple
- should be natural compilation target for interesting problems
- should have community focusing on strong algorithms
- should capture the spirit of a complexity class

for contrast, compare Generalized Tic-Tac-Toe
Reason 2: Planning is Clean and Simple

Planning is a clean and simple problem.

- I would expect some opposition here.
- Isn’t planning an awfully specific problem compared to SAT, MaxSAT, QBF, . . . ?
- **Claim**: the underlying abstract problem isn’t, but there is historical baggage obscuring this
- What is the *essence* of planning?
Capturing the Essence (1)

Cleaning up our previous definition:

- **finite-domain state variables:**
  \( x \in \{a, b, c\}, \ y \in \{a, b\}, \ z \in \{a, b, c\} \)

- **initial state:**
  \( \{x \mapsto a, \ y \mapsto a, \ z \mapsto a\} \)

- **goal:**
  \( \{x \mapsto c, \ z \mapsto b\} \)

- **actions:**
  \( a_1 : \quad x = a, \ y = a \quad \rightarrow \quad y := b, \ z := c \)
  \( a_2 : \quad \ldots \)
  \( \ldots \)

**Question:** is there a sequence of actions transforming the initial state to a state consistent with the goal?
Capturing the Essence (2)

Observations:

- no generality lost by restricting to Boolean state variables $V$
- no generality lost by restricting to unique goal state
- no generality lost by fixing a “canonical” initial and goal state, as long as they are different
Capturing the Essence (3)

A simpler description:

- propositional variables $V$
- actions:
  - $a_1 : \ x = T, \ y = T \ \rightarrow \ \ y := F, \ z := T$
  - $a_2 : \ldots$
  - \ldots

**Question:** is there a sequence of actions transforming

\[ F^V := \{ v \mapsto F \mid v \in V \} \] to \[ T^V := \{ v \mapsto T \mid v \in V \}? \]
Capturing the Essence (4)

- OK, this looks simpler!
- But it still looks very planningish with all these actions and states...
- What is the abstract purpose of “actions”?
- Specifying how one can transition from one truth assignment (state) to another.
- We can express this more generically.
Capturing the Essence (5)

A simpler description:

- propositional variables $V$
- a transition relation for $V$, expressed as a propositional formula $\varphi(V, V')$

**Question:** is there a sequence of truth assignments $I_0, \ldots, I_n$ with

- $I_0 = F^V$,
- $I_n = T^V$, and
- $\varphi(I_k, I'_{k+1})$ is satisfied for all $0 < k < n$?
Reason 2: Planning is Clean and Simple

Planning is a clean and simple problem.

It’s **satisfiability** + transitive closure.
Reason 3: **Planning is a Natural Compilation Target**

Planning is a natural compilation target for interesting problems.
Reason 3: Planning is a Natural Compilation Target

Planning is a natural compilation target for interesting problems.

Some problems that have been tackled by compilation to planning:

- formal verification (software, hardware, protocols)
- network security analysis
- diagnosis of discrete-event systems
- sentence generation
- narrative generation
- music composition
- business process model verification
- level design for computer games
- phylogenetic path analysis
- molecular synthesis
Reason 3: Planning is a Natural Compilation Target

Planning is a natural compilation target for interesting problems.

See also:
AAAI 2012 Workshop on Problem Solving using Classical Planners
Reason 4: Strong Community with Algorithmic Focus

Planning has a strong community of researchers pushing the limits of problem solvers (planners).

- regular competitions with tremendous progress
- rich and growing benchmark suite
- separation of problem solvers (algorithm developers) and customers (users expressing their problems as planning tasks)
Reason 5: Planning Captures the Spirit of PSPACE

Planning is a very natural way to capture PSPACE.

What do I mean by this?

- SAT is well-suited to express the computations possible in NP: that’s why the Cook-Levin theorem is usually presented with reductions to SAT, not to Minesweeper
- Planning is similarly well-suited to express PSPACE: if you understand Cook-Levin, you can do the reduction from generic problems in PSPACE to Planning
Reduce $A \in \text{PSPACE}$ to planning:

Consider TM $M$ solving $A$ with polynomial space bound $p(n)$.

- TM states: $Q$
- initial state $q_0$
- accepting state $q_{\text{accept}}$
- tape alphabet $\Gamma$

$\rightarrow$ for input $w$, may visit tape cells

\[ \text{Pos}(w) = \{-p(|w|), \ldots, p(|w|)\} \]
Reducing a Generic PSPACE Problem to Planning (2)

Map to planning task with:

- **state variables:**
  - `tm_state` with domain `Q`
  - `head` with domain `Pos(w)`
  - `tape_p` for each `p ∈ Pos(w)` with domain `Γ`

- **initial state:** describes initial TM configuration

- **goal:** `\{tm\_state \mapsto q_{accept}\}`

- **action** for each TM rule `q, a \rightarrow q', a', \delta`
  and tape position `p ∈ Pos(w)`:  

\[
\begin{align*}
\text{tm\_state} &= q, \text{head} = p, \text{tape}_p = a \\
\rightarrow \text{tm\_state} &:= q', \text{head} := p + \delta, \text{tape}_p := a'
\end{align*}
\]

\(\Rightarrow\) planning task has solution iff `A` accepts `w`

\(\Rightarrow\) `A \leq_p` Planning
How Does Planning Fit into the Beyond NP Landscape?
Beyond NP: PSPACE

Who rules PSPACE then?

- **this talk**: planning as a compilation target for PSPACE
- **next talk**: QBF as a compilation target for PSPACE

What gives?
I am not convinced that “one problem to rule them all” is the correct approach for PSPACE.

People have tried to express planning as QBF: so far, this has not been a remotely viable approach.

Conversely, compiling QBF to planning is not remotely viable with the current planning algorithms.

Is there something about PSPACE that explains this?
The Many Faces of PSPACE (1)

PSPACE has several faces:

- **DPSPACE**: DTM$s + polynomial space
- **NPSPACE**: NTM$s + polynomial space
- **APTIME**: ATM$s + polynomial time

We know \( \text{DPSPACE} = \text{NPSPACE} = \text{APTIME} \).

For different problems in PSPACE, different faces are natural.
Similarly in descriptive complexity theory (DCT):

- **PSPACE** has multiple competing characterizations, mixing various flavors of second-order quantification and transitive closure

- unlike classes like **NP** or **Σ²_p** with one most prominent characterization:
  - **NP**: \( \exists X \varphi[X] \)
  - **Σ²_p**: \( \exists X \forall Y \varphi[X, Y] \)

\((X, Y: \text{second-order variables}; \varphi: \text{first-order formula})\)
Concluding statement:

- For some problems in PSPACE, a description in terms of existential quantification plus transitive closure is natural.
- For other problems in PSPACE, a description in terms of alternating quantifiers is natural.
- Shoehorning a problem of one type to a representation of the other type is a bad idea.

Caveat: tomorrow’s solvers (on either side) might bridge this gap.

But today we are far away from this.
Thank you for your attention!