DISSETRATION:

Advances in Answer Set Planning

Dipl.-Ing. Axel Polleres
axel@kr.tuwien.ac.at

supervised by
O.Univ.Prof. Dipl.-Ing. Dr.techn. Thomas Eiter

Institut für Informationssysteme, Abteilung für Wissensbasierte Systeme
Overview – Contributions

- Preliminaries
Overview – Contributions

- Preliminaries
- Novel Declarative Planning Language $\mathcal{K}^c$
  - Syntax, Semantics
  - Complexity of Planning in $\mathcal{K}^c$
Overview – Contributions

- Preliminaries
- Novel Declarative Planning Language $\mathcal{K}^c$
  - Syntax, Semantics
  - Complexity of Planning in $\mathcal{K}^c$
- Problem Solving in Answer Set Programming (ASP) by “Guess and Check”
Overview – Contributions

- Preliminaries
- Novel Declarative Planning Language $\mathcal{K}^c$
  - Syntax, Semantics
  - Complexity of Planning in $\mathcal{K}^c$
- Problem Solving in Answer Set Programming (ASP) by “Guess and Check”
- Translation of Planning in $\mathcal{K}^c$ to ASP
Overview – Contributions

- Preliminaries
- Novel Declarative Planning Language $\mathcal{KC}$
  - Syntax, Semantics
  - Complexity of Planning in $\mathcal{KC}$
- Problem Solving in Answer Set Programming (ASP) by “Guess and Check”
- Translation of Planning in $\mathcal{KC}$ to ASP
- Ready-to-Use Planning System $\mathcal{DLV^K}$
  - Implementation
  - Experiments
Overview – Contributions

- Preliminaries
- Novel Declarative Planning Language $\mathcal{K}^c$
  - Syntax, Semantics
  - Complexity of Planning in $\mathcal{K}^c$
- Problem Solving in Answer Set Programming (ASP) by “Guess and Check”
- Translation of Planning in $\mathcal{K}^c$ to ASP
- Ready-to-Use Planning System $\text{DLV}^\mathcal{K}$
  - Implementation
  - Experiments
- Application: Planning for MAS-Monitoring
Preliminaries:

Planning Problem: Find a sequence of actions to bring an agent from an initial state to a goal state

Input: A set of actions (preconditions, effects);
Fluents (state variables) and their initial values and goal values

Output: Sequence of action sets (discrete notion of time)
Preliminaries:

Planning Problem: Find a sequence of actions to bring an agent from an initial state to a goal state

- **Input:** A set of actions (preconditions, effects); Fluents (state variables) and their initial values and goal values

- **Output:** Sequence of action sets (discrete notion of time)

Classical Planning (complete knowledge, deterministic actions)

- **Actions:** move(B,L)
- **Fluents:** on(b,table), on(c,a), on(a,table), occupied(a), ...

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>c</td>
<td>a</td>
</tr>
</tbody>
</table>
```

```
<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>b</td>
<td>a</td>
</tr>
</tbody>
</table>
```
Preliminaries:

Planning Problem: Find a sequence of actions to bring an agent from an initial state to a goal state

Input: A set of actions (preconditions, effects); Fluents (state variables) and their initial values and goal values

Output: Sequence of action sets (discrete notion of time)

Classical Planning (complete knowledge, deterministic actions)

Actions: move(B,L)
Fluents: on(b,table), on(c,a), on(a,table), occupied(a), ...

Non-Classical Planning (Conformant Plans, Conditional Plans, ...)
Planning and Action Languages

Existing formal Languages: \textsc{strips}, \textsc{adl}, \textsc{pddl}, \mathcal{A}, \mathcal{C}, \ldots

Here: Novel planning language $\mathcal{K}^c$:

$\mathcal{K}^c$– Features:

- A relative of action languages $\mathcal{A}$ (Gelfond & Lifschitz, 1993) and $\mathcal{C}$ (Giunchiglia & Lifschitz, 1998)
Planning and Action Languages

Existing formal Languages: STRIPS, ADL, PDDL, $A$, $C$, ...

Here: Novel planning language $\mathcal{K}^c$:

$\mathcal{K}^c$ – Features:

- A relative of action languages $A$ (Gelfond & Lifschitz, 1993) and $C$ (Giunchiglia & Lifschitz, 1998)
- Incomplete states (“knowledge states”)
Planning and Action Languages

Existing formal Languages: STRIPS, ADL, PDDL, A, C, ... 

Here: Novel planning language $K^c$:

$K^c$ – Features:

- A relative of action languages $A$ (Gelfond & Lifschitz, 1993) and $C$ (Giunchiglia & Lifschitz, 1998)
- Incomplete states ("knowledge states")
- Default (nonmonotonic) negation and strong (classical) negation
Planning and Action Languages

Existing formal Languages: STRIPS, ADL, PDDL, \( \mathcal{A} \), \( \mathcal{C} \), ... 

Here: Novel planning language \( \mathcal{K}^c \): 

\( \mathcal{K}^c \) – Features:

- A relative of action languages \( \mathcal{A} \) (Gelfond & Lifschitz, 1993) and \( \mathcal{C} \) (Giunchiglia & Lifschitz, 1998)
- Incomplete states ("knowledge states"")
- Default (nonmonotonic) negation and strong (classical) negation
- Nondeterministic action effects
Planning and Action Languages

Existing formal Languages: STRIPS, ADL, PDDL, \( \mathcal{A} \), \( \mathcal{C} \), \ldots

Here: Novel planning language \( \mathcal{K}^c \):

\( \mathcal{K}^c \) – Features:

- A relative of action languages \( \mathcal{A} \) (Gelfond & Lifschitz, 1993) and \( \mathcal{C} \) (Giunchiglia & Lifschitz, 1998)
- Incomplete states (“knowledge states”)
- Default (nonmonotonic) negation and strong (classical) negation
- Nondeterministic action effects
- Action costs
$\mathcal{K}^c$ Planning Domains and Problems

**Background Knowledge $\Pi$:** A logic program $\Pi$ with a single model, (answer set) defining type information and static knowledge.

**$\mathcal{K}$ Action Description $AD$:**

- **fluenets**: $D_F$  % fluent declarations
- **actions**: $D_A$  % action type declarations
- **always**: $C_R$  % causation rules + exec. cond’s
- **initially**: $C_I$  % initial state constraints

**$\mathcal{K}$ Planning Domain:** $\langle \Pi, AD \rangle$

**$\mathcal{K}$ Planning Problem:** additional goal

- **goal**: $G?^{(i)}$  ground literal(s) $G$; plan length $i \geq 0$. 
Blocks world in $\mathcal{K}^c$

initial: $b\ c\ a$
goal: $c\ b\ a$

Background knowledge

$\Pi = \{ \text{block}(a). \text{block}(b). \text{block}(c).$
\[\text{location}(\text{table}).\]
\[\text{location}(L) \leftarrow \text{block}(L).\} \]

(Logic Program which has a single model - set of “invariant” facts)
Blocks world: $\mathcal{K}^C$ Problem description

fluent: on(B,L) requires block(B), location(L).
occupied(B) requires location(B).

actions: move(B,L) requires block(B), location(L) costs 1.
Blocks world: $K^C$ Problem description

fluent:  
- on(B,L) requires block(B), location(L).
- occupied(B) requires location(B).

action:  
- move(B,L) requires block(B), location(L) costs 1.

always:  
- executable move(B,L) if not occupied(B), not occupied(L), B<>L.
- caused on(B,L) after move(B,L).
- caused $\neg$on(B,L1) after move(B,L), on(B,L1), L<>L1.
- caused occupied(B) if on(B1,B), block(B).

inertial on(B,L). % Explicit frame axioms!
noConcurrency. % Optionally, parallel actions!
Blocks world: $K^C$ Problem description

fluent: on(B,L) requires block(B), location(L).
occupied(B) requires location(B).

actions: move(B,L) requires block(B), location(L) costs 1.

always: executable move(B,L) if not occupied(B), not occupied(L), B<>L.
caused on(B,L) after move(B,L).
caused ¬on(B,L1) after move(B,L), on(B,L1), L<>L1.
caused occupied(B) if on(B1,B), block(B).
inertial on(B,L). % Explicit frame axioms!
noConcurrency. % Optionally, parallel actions!

initially: on(a,table). on(b,table). on(c,a).

goal: on(c,b), on(b,a), on(a,table)? (3)
Blocks world: $K^c$ Problem description

fluents:  
on(B,L) requires block(B), location(L).
occupied(B) requires location(B).

actions:  
move(B,L) requires block(B), location(L) costs 1.

always:  
executable move(B,L) if not occupied(B), not occupied(L), B<>L.
caused on(B,L) after move(B,L).
caused ~on(B,L1) after move(B,L), on(B,L1), L<>L1.
caused occupied(B) if on(B1,B), block(B).
inertial on(B,L). % Explicit frame axioms!
noConcurrency. % Optionally, parallel actions!

initially: on(a,table). on(b,table). on(c,a).
goal: on(c,b), on(b,a), on(a,table)? (3)

Intuitively: Feasible plan is

move(c,table); move(b,a); move(c,b) COSTS: 3
Incomplete Knowledge:

Initially: total on(d,L).
caused ¬on(d,c).

% State Axioms:
caused false if on(B,L), on(B,L1), L<>L1.
caused false if on(B1,B), on(B2,B), block(B), B1<>B2.

goal: on(a,c), on(c,d), on(d,b), on(b,table)? (4)
Incomplete Knowledge:

initially: \texttt{total on(d,L)}.
caused \texttt{\neg on(d,c)}.

State Axioms:
caused false if \texttt{on(B,L), on(B,L1), L<>L1}.
caused false if \texttt{on(B1,B), on(B2,B), block(B), B1<>B2}.

\textbf{Goal:} \texttt{on(a,c), on(c,d), on(d,b), on(b,table)}? (4)

Feasible plans:
\texttt{move(c,d); move(a,c); (no action); (no action)} COSTS: 2
\texttt{move(d,c); move(d,b); move(c,d); move(a,c)} COSTS: 4
Semantics of $\mathcal{K}^c$ – Plans

Multi-valued transition function $t(s, A)$, LP-based (Answer Sets!)

**Optimistic plans:**

- $S_0 \rightarrow A_1 \rightarrow S_1 \rightarrow \cdots \rightarrow A_n \rightarrow S_n$
- Goal

**Secure plans:**

- $S_0 \rightarrow A_1 \rightarrow S_1 \rightarrow \cdots \rightarrow A_n \rightarrow S_n$
- Goal

**Optimal plans:** plans with lowest cost

**Admissible plans:** plans which stay within fixed cost limit
### $\mathcal{K}^c$ Complexity

<table>
<thead>
<tr>
<th>PD</th>
<th>plan length $i$ in query $q = \text{Goal} \ ? (i)$</th>
<th>fixed (=constant)</th>
<th>arbitrary</th>
</tr>
</thead>
<tbody>
<tr>
<td>general</td>
<td>NP / $\Pi^P_2$ / $\Sigma^P_3$ -complete</td>
<td>PSPACE / $\Pi^P_2$ / NEXPTIME -complete</td>
<td></td>
</tr>
<tr>
<td>proper</td>
<td>NP / co-NP / $\Sigma^P_2$ -complete</td>
<td>PSPACE / co-NP / NEXPTIME -complete</td>
<td></td>
</tr>
</tbody>
</table>

Complexity Results for Optimistic Planning / Security Checking / Secure Planning in $\mathcal{K}$ (Propositional Case)

Planning with Action costs:

**Computing** optimal optimistic/secure plans is $F\Delta^P_2$-complete/$F\Delta^P_4$-complete.

Answer Set Programming (with weak constraints) can be used to solve some of these tasks!
Answer Set Programming (ASP)

ASP with weak constraints is capable of solving problems beyond NP!
(conformant planning for proper domains, general secure checking)

$\Delta^p_3 = \Sigma^P_2$

$\Sigma^P_2 = \text{NP} = \text{NP}^{\text{co-NP}}$

NP
SAT, normal LP, HCF–LP

co–NP
UnSAT, ...

- Solvers for NP problems: Smodels, SAT-Solvers, ...
- Solvers for $\Sigma^P_2$ problems: DLV, GnT, ...
- Solvers for $\Delta^P_3$ problems: DLV with weak constraints ...
Answer Set Programming (ASP)

- function-free, disjunctive Logic Programs, set of rules:

\[ h_1 \lor \ldots \lor h_l :\sim b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots \text{not } b_n. \]

- Semantics: Answer Sets Semantics for nonmonotonic logic programs (Gelfond & Lifschitz, 1991), minimal “stable” models

- Extension: weak constraints (Buccafurri et.al., 1999):

\[ :\sim b_1, \ldots, b_m, \text{not } b_{m+1}, \ldots \text{not } b_n. [C] \]

- Semantics: Optimal Answer Sets (with minimal violation costs)
Problem Solving in ASP:

“Guess and Check” Paradigm: a Simple Example:

\[
\text{col}(X, r) \lor \text{col}(X, g) \lor \text{col}(X, b) \leftarrow \text{node}(X). \] \quad \text{Guess}
\]

\[\leftarrow \text{edge}(X, Y), \text{col}(X, C), \text{col}(Y, C). \] \quad \text{Solution Check}

**Input:** A graph represented by node(\_\_) and edge(\_, \_\_).

**Problem:** Assign a color to all nodes such that adjacent nodes always have different colors.

**NP-complete problem!**

ASP is well suited for solving search problems with a finite search space! Efficient solvers (DLV, smodels, . . . ) exist!
Beyond NP: 2QBFs

\[ \Psi = \exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_n \psi \]

\[ \psi = d_1 \lor \cdots \lor d_k \]

\[ d_i = a_{i,1} \land \cdots \land a_{i,l_i} \text{ and } |a_{i,j}| \in \{x_1, \ldots, x_m, y_1, \ldots, y_n\} \]

Compute an assignment to \( x_1, \ldots, x_m \) such that \( \Psi \) is true
Beyond NP: 2QBFs

\[ \Psi = \exists x_1 \ldots \exists x_m \forall y_1 \ldots \forall y_n \psi \]

\[ \psi = d_1 \lor \ldots \lor d_k \]

\[ d_i = a_{i,1} \land \ldots \land a_{i,l_i} \text{ and } |a_{i,j}| \in \{x_1, \ldots, x_m, y_1, \ldots, y_n\} \]

Compute an assignment to \( x_1, \ldots, x_m \) such that \( \Psi \) is true

\[
\begin{align*}
x_1 & \lor nx_1 \ldots x_m \lor nx_m. \\
y_1 & \lor ny_1 \ldots y_n \lor ny_n. \\
\text{sat} & := a_{1,1}, \ldots, a_{1,l_1}. \\
& \vdots \\
\text{sat} & := a_{k,1}, \ldots, a_{k,l_k}. \\
y_1 & := \text{sat}. \ldots y_n := \text{sat}. \\
ny_1 & := \text{sat}. \ldots ny_n := \text{sat}. \\
& := \text{not sat}. 
\end{align*}
\]

Check part uses “saturation” technique!
Integrate “Guess” and “Check”

Problems:

- Integrated $\Sigma^P_2$ programs often hard to find,
- “Guess” and “Check” structure hard to see.
- $\Sigma^P_2$ encodings use unintuitive “saturation” techniques.
Integrate “Guess” and “Check”

Problems:

- Integrated $\Sigma^P_2$ programs often hard to find,
- “Guess” and “Check” structure hard to see.
- $\Sigma^P_2$ encodings use unintuitive “saturation” techniques

Solution: Interleaved Computation!

- Separate programs $\Pi_{guess}$ and $\Pi_{check}$
- Compute solutions interleaved, i.e. compute all solutions $S$ to $\Pi_{guess}$ and only accept those where $\Pi_{check}(S)$ has no answer set.
Integrate “Guess” and “Check”

Problems:

- Integrated $\Sigma_2^P$ programs often hard to find,
- “Guess” and “Check” structure hard to see.
- $\Sigma_2^P$ encodings use unintuitive “saturation” techniques

Solution: Interleaved Computation!

- Separate programs $\Pi_{guess}$ and $\Pi_{check}$
- Compute solutions interleaved, i.e. compute all solutions $S$ to $\Pi_{guess}$ and only accept those where $\Pi_{check}(S)$ has no answer set.
- Thesis provides automatic method for combining these programs!
Based on this method, we define ASP Translations for:

- optimistic planning
- general/proper secure checking
- proper secure planning
Based on this method, we define ASP Translations for:

- optimistic planning
- general/proper secure checking
- proper secure planning

Action costs (optimal/admissible planning):

- Extend these translations by weak constraints feature
Plan Generator: Computes Optimistic Plans
Plan Checker: Checks Optimistic Plans for Security
Planning for Multi-Agent Monitoring

Idea:

- Model collaborative Behavior in an MAS in $\mathcal{K}^c$, and
- derive valid messaging protocols from plans.
Planning for Multi-Agent Monitoring

Idea:
- Model collaborative Behavior in an MAS in $\mathcal{K}^c$, and
- derive valid messaging protocols from plans.

Further interesting applications:
- Optimal Route planning with exceptional, time-dependent costs
- Cheapest among the shortest plans, Shortest among the cheapest plans
- Conformant planning examples from the literature (SQUARE, Bomb in Toilet)
Conclusions & Outlook

Expressive planning language based on Logic Programming (Knowledge Representation!)

Efficient planning system based on Answer Set Techniques is feasible:
- DLV

Encouraging Results (Experiments!)
Exploiting the full potential of ASP power (encodings), problems beyond SAT Solvers!

Further Work: Reactive Planning!
Conclusions & Outlook

Expressive planning language based on Logic Programming (Knowledge Representation!)

Efficient planning system based on Answer Set Techniques is feasible: $DLV^K$
Conclusions & Outlook

- Expressive planning language based on Logic Programming (Knowledge Representation!)

- Efficient planning system based on Answer Set Techniques is feasible: \( \text{DLV}^K \)

- Encouraging Results (Experiments!)
Conclusions & Outlook

- Expressive planning language based on Logic Programming (Knowledge Representation!)

- Efficient planning system based on Answer Set Techniques is feasible: $\text{DLV}^K$

- Encouraging Results (Experiments!)

- Exploiting the full potential of ASP power ($\Sigma_2^p$ encodings), problems beyond SAT Solvers!
Conclusions & Outlook

Expressive planning language based on Logic Programming (Knowledge Representation!)

Efficient planning system based on Answer Set Techniques is feasible: \( DLV^K \)

Encouraging Results (Experiments!)

Exploiting the full potential of ASP power \( (\Sigma_2^P \text{ encodings}) \), problems beyond SAT Solvers!

Further Work: Reactive Planning!
Selected Publications

T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres.  
*A Logic Programming Approach to Knowledge-State Planning: Semantics and Complexity.*  

T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres.  
*A Logic Programming Approach to Knowledge-State Planning, II: the DLV$^\mathcal{C}$ System.*  

T. Eiter, W. Faber, N. Leone, G. Pfeifer, and A. Polleres.  
*Answer Set Planning under Action Costs.*  

*Monitoring agents using planning.*  
German Conference on Artificial Intelligence (KI2003), 2003.

T. Eiter and A. Polleres.  
*Towards Automated Integration of Guess and Check Programs in Answer Set Programming.*  
Accepted for 7th International Conference on Logic Programming and Nonmonotonic Reasoning (LPNMR-7).
Semantics of $\mathcal{K}^c$ – Transitions

Transition-based semantics: “legal” transitions $\langle s_0, A, s_1 \rangle$

caused $fl$ if $\text{Cond1}$ after $\text{Cond2}$

\[
\begin{align*}
&s_0 \rightarrow A \rightarrow s_1
\end{align*}
\]

A \ldots set of actions (executable)

- $\text{Cond2}$ is evaluated in $s_0$
- $fl$ and $\text{Cond1}$ are evaluated in $s_1$

Define new state $s_1$ by a non-monotonic logic program of rules

\[
fl :- \text{Cond1}
\]

Remark:
e.g., transitive closure easily expressed (LP-flavored semantics of $\mathcal{K}$)