Generating None-Plans in Order to Find Plans

Michał Knapik*

IPI PAN, 29.II.2016

*a joint work with Wojciech Penczek and Artur Niewiadomski
Outline

1. Introduction
2. Planning in PlanICS
3. Simplified Planning Domain
4. Plans and None-plans
5. Synthesis of None-Plans
6. Applying None-Plans to Find Plans
7. Experimental Results
Main Contributions

- A (new?) method for improving efficiency of algorithms solving hard problems,
- A (new?) reduction method for planning,
- Application of the results in the tool PlanICS.
Related Work

- Planning methods and tools: OWLS-Xplan, OWLS-MX, WSMO, PDDL3, PlanICS, . . . ,
- Abstraction methods [Cousot, . . . ],
- Partial order reductions [Valmari, Peled, Godefroid, . . . ],
- Symmetry reductions [Clarke, Emerson, Jha, Sistla, . . . ],
- CEGAR – Counterexample Guided Abstraction [Clarke],
- . . . and others.
General idea – intuition

- **D** – a domain to find a plan (problem is NP-complete),
- **D’** – an abstract domain in which finding a plan is easy,
- a plan in **D’** does not need to correspond to a plan in **D**,
- a none-plan in **D’** corresponds to a none-plan in **D**,
- find (the) none-plans in **D’**,
- prune **D** from (the) none-plans of **D’**,
- search for (the) plans in **D** pruned.
Application to planning in PlanICS

- Given an **ontology of object types and services**, 
- Given a user query: \((\text{initial worlds}, \text{final worlds})\), 
- A **world** – a set of **objects** (object has a **type** and **attributes**),
- A **service**: \((\text{in}, \text{inout}, \text{out}, \text{pre}, \text{post})\), where \(\text{in}, \text{inout}, \text{out}\) are sets of objects,
- **pre** – a bool. form. over **object attributes** of \(\text{in}\) and \(\text{inout}\), 
- **post** – a bool. form. over **object attributes** of \(\text{inout}\) and \(\text{out}\).

**Task:** Find all plans from some **initial** to some **final world**.

\((This\ \text{problem}\ \text{is}\ \text{NP-complete}.)\)
Service composition in PlanICS

Initial world

pre\textsubscript{A}

preconditions on object attributes

postconditions on object attributes

Service A → World 0 → Service B → World 1 → Service C

post\textsubscript{A}
pren\textsubscript{B}
pren\textsubscript{C}

post\textsubscript{B}

post\textsubscript{C}

Final world

Planning – composition of services (a huge number of plans)
Service composition in PlanICS

Planning – composition of services (a huge number of plans)
Simplifying the planning domain

Idea – simplify services and worlds:

- the simplified objects do not have attributes,
- a simplified world – a multiset of objects,
- a simplified service – \((\text{precondition}, \text{effect})\),
- precondition – a multiset of objects (objects required),
- effect – a multiset of objects (new objects added).

- \(B\) – a set of services, \(B'\) – the set of simplified services,
- Fact: If \(B'\) cannot be composed into a plan, then \(B\) cannot be composed into a plan,
- Goal: synthesize constraints of non-composability.
(Simplified) Planning Domain $\mathcal{P} = (\mathcal{W}_\mathcal{H}, F_I, F_G, Act)$:

- $\mathcal{W}_\mathcal{H} \subseteq \mathbb{N}^n$ – a set of abstract worlds (multisets),
- $F_I, F_G \subseteq \mathcal{W}_\mathcal{H}$ – initial, final worlds,
- $Act$ – a set of actions (simplified services).

where $n$ is the number of all types of the objects.

For each $act \in Act$:

- $pre(act)$ – precondition of act,
- $eff(act)$ – effect of act.

$pre(act), eff(act) \in \mathbb{N}^n$.

Action $act \in Act$ is enabled in $\omega \in \mathcal{W}_\mathcal{H}$ iff $pre(act) \leq \omega$ and the results of firing $act$: $\omega \xrightarrow{act} \omega + eff(act)$
Given $\mathcal{P} = (\mathcal{W_H}, F_I, F_G, \text{Act})$, $B \subseteq \text{Act}$

- $\pi \in \Pi(\omega, B, \omega')$ iff

$$\pi = \omega_0 \xrightarrow{\text{act}_1} \omega_1 \xrightarrow{\text{act}_2} \ldots \xrightarrow{\text{act}_{n-1}} \omega_{n-1} \xrightarrow{\text{act}_n} \omega_n$$

where $\omega_0 = \omega$, $\omega_n \geq \omega'$, and $\{\text{act}_1, \ldots, \text{act}_n\} \subseteq B$

- $\bigcup_{\omega_I \in F_I} \bigcup_{\omega_F \in F_G} \Pi(\omega_I, B, \omega_F)$ – the plans over $B$

Each plan starts from an initial world and its last world covers a final world.
Exemplary planning domain

Actions:

- **makeVehicle**:
  needs nothing, builds vehicle

- **makeCar**:
  needs vehicle, builds car

- **makeBoat**:
  needs vehicle, builds boat

- **makeAmphibian**:
  needs boat and car, builds amphibian

- **tinker**:
  needs amphibian and car, builds two amphibians
Exemplary planning domain, ct’d

Order of the objects: 
(Vehicle, Car, Boat, Amphibian)

- \text{makeAmphibian}: needs boat and car, builds amphibian

\[
\text{pre}(\text{makeAmphibian}) = (1, 0, 1, 0) + (1, 1, 0, 0) = (2, 1, 1, 0)
\]

\[
\text{eff}(\text{makeAmphibian}) = (1, 1, 1, 1)
\]
Exemplary planning domain, ct’d

Order of the objects:
(Vehicle, Car, Boat, Amphibian)

- `makeAmphibian`:
  needs boat and car, builds amphibian

\[
\text{pre}(\text{makeAmphibian}) = (1, 0, 1, 0) + (1, 1, 0, 0) = (2, 1, 1, 0)
\]

\[
\text{eff}(\text{makeAmphibian}) = (1, 1, 1, 1)
\]
Exemplary planning domain, ct’d

Order of the objects: 
(Vehicle, Car, Boat, Amphibian)

- **makeAmphibian**: 
  needs boat and car, builds amphibian

\[
\text{pre}(\text{makeAmphibian}) = (1, 0, 1, 0) + (1, 1, 0, 0) = (2, 1, 1, 0)
\]

\[
\text{eff}(\text{makeAmphibian}) = (1, 1, 1, 1)
\]
Exemplary planning domain, ct’d

Order of the objects:

(Vehicle, Car, Boat, Amphibian)

- **makeAmphibian:**
  needs boat and car, builds amphibian

\[
\text{pre}(\text{makeAmphibian}) = (1, 0, 1, 0) + (1, 1, 0, 0) = (2, 1, 1, 0)
\]

\[
\text{eff}(\text{makeAmphibian}) = (1, 1, 1, 1)
\]
Exemplary planning domain, ct’d

Actions:

- \( \text{pre}(\text{makeVehicle}) = (0, 0, 0, 0) \)
  \( \text{eff}(\text{makeVehicle}) = (1, 0, 0, 0) \)

- \( \text{pre}(\text{makeCar}) = (1, 0, 0, 0) \)
  \( \text{eff}(\text{makeCar}) = (1, 1, 0, 0) \)

- \( \text{pre}(\text{makeBoat}) = (1, 0, 0, 0) \)
  \( \text{eff}(\text{makeBoat}) = (1, 0, 1, 0) \)

- \( \text{pre}(\text{makeAmphibian}) = (2, 1, 1, 0) \)
  \( \text{eff}(\text{makeAmphibian}) = (1, 1, 1, 1) \)

- \( \text{pre}(\text{tinker}) = (2, 2, 1, 1) \)
  \( \text{eff}(\text{tinker}) = (2, 2, 2, 2) \)

\( \omega_I = (0, 0, 0, 0) \) (one initial world)
\( \omega_F = (0, 0, 0, 1) \) (one final world)
Classifying actions

$V_{\text{max}}$ - the largest number occurring in $\text{pre}(\text{act})$ for $\text{act} \in \text{Act}$.

$\text{enact}(A) = \{ \text{act} \in \text{Act} | \sum_{\text{act}' \in A} V_{\text{max}} \cdot \text{eff}(\text{act}') \geq \text{pre}(\text{act}) \}$.

all actions that can be enabled by firing actions from $A \subseteq \text{Act}$,

$\omega \in \mathcal{W}_{\mathcal{H}}, i > 0$
- $G^\omega_0 = \{ \text{act} \in \text{Act} | \text{pre}(\text{act}) \leq \omega \}$ – the actions enabled in $\omega$,
- $G^\omega_{i+1} = \text{enact}(G^\omega_i)$ – the actions enabled in $i$–th step
- $H^\omega_0 = G^\omega_0$,
- $H^\omega_{i+1} = G^\omega_{i+1} \setminus G^\omega_i$ – the actions newly enabled in $i$–th step.
Classifying actions, ct’d

\[ \omega, \omega' \in \mathcal{W}_H \]

\[ k_{\text{goal}}(\omega, \omega') = \min \left( \{ k \in \mathbb{N} \mid \sum_{\text{act} \in G_k} V_{\max} \cdot \text{eff}(\text{act}) \geq \omega' \} \right) \]

the minimal step at which greedily fired actions cover \( \omega' \).

**Lemma A**

- \( k_{\text{goal}}(\omega, \omega') < \infty \) iff \( \Pi(\omega, \text{Act}, \omega') \neq \emptyset \),
- \( k_{\text{goal}}(\omega, \omega') \) can be computed in time \( O(|\text{Act}|^2 \cdot n) \).

Planning in \( \mathcal{P} \) is easy.
Classifying actions, cont’d

\[ \omega_I \in F_I, \omega_F \in F_G, \text{ klimit}(\omega_I) = \min(\{ k \in \mathbb{N} \mid H_{k}^{\omega_I} = \emptyset \}) \]

- \( \mathcal{E} = \text{Act} \setminus G_{\text{klimit}(\omega_I)}^{\omega_I} \) – **useless** actions can’t be enabled
- \( \mathcal{G} = G_{\text{kgoal}(\omega_I,\omega_F)}^{\omega_I} \) – **sufficient** actions can cover goal
- \( \mathcal{R} = \{ \text{act} \in G_{\text{klimit}(\omega_I)}^{\omega_I} \mid \text{pre}(\text{act}) \geq \omega_F \} \) – **redundant** actions
- \( \mathcal{T} = G_{\text{klimit}(\omega_I)}^{\omega_I} \setminus (G_{\text{kgoal}(\omega_I,\omega_F)}^{\omega_I} \cup \mathcal{R}) \) – **potentially** useful acts
Classifying actions, cont’d

Lemma B
Let $A \subseteq \text{Act}$. If there is a plan over $A$, then $A$ contains at least one element from $H_i^{\omega_l}$ for all $0 \leq i \leq k_{\text{goal}}(\omega_l, \omega_F)$

First easy reductions:
- throw away redundant (e.g., tinker) and useless actions,
- block all sets of actions that do not satisfy Lemma B.

More reductions: consider none-plans.
None-plans

\( A \subseteq \text{Act}, \omega, \omega' \in \mathcal{W}_H \)

\[ \mathcal{Z}(\omega, A, \omega') := \{ B \subseteq A \mid \Pi(\omega, B, \omega') = \emptyset \} \]

None-plan: a set of actions \( B \), which is not a support of a plan starting at \( \omega \) and covering \( \omega' \).

\( \Pi(\omega) := \{ \omega' \mid \|\omega'\| = 1 \land \omega \geq \omega' \} \) – unitary coord. vects. of \( \omega \)

e.g., \( \Pi((2, 1, 1, 0)) = \{ (1, 0, 0, 0), (0, 1, 0, 0), (0, 0, 1, 0) \} \)
Characterisation of none-plans

Theorem

\[ \mathcal{Z}(\omega, A, \omega') = \bigcup_{\omega'' \in \Pi(\omega')} \bigcap_{\omega \not\geq \omega'' \text{ and } \text{eff}(act) \geq \omega''} \left( \mathcal{D}(\omega, A, act) \cup 2^A \setminus \{act\} \right) \]

where \( \mathcal{D}(\omega, A, act) = \{B \cup \{act\} \mid B \in \mathcal{Z}(\omega, A \setminus \{act\}, \text{pre}(act))\} \)
Theorem

\[ \mathcal{Z}(\omega, A, \omega') = \bigcup_{\omega''\in \mathcal{I}(\omega')} \bigcap_{\text{eff}(\text{act}) \geq \omega''} (\mathcal{D}(\omega, A, \text{act}) \cup 2^A\setminus\{\text{act}\}) \]

where \( \mathcal{D}(\omega, A, \text{act}) = \{B \cup \{\text{act}\} \mid B \in \mathcal{Z}(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))\} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \), take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered for each action \( \text{act} \) that could cover \( \omega'' \) when fired, ensure that \( \text{act} \) cannot be enabled and take it or throw \( \text{act} \) away.
Characterisation of none-plans, ct’d

Theorem

\[ Z(\omega, A, \omega') = \bigcup_{\omega'' \in \Pi(\omega')} \bigcap_{\omega \not\geq \omega''} (D(\omega, A, \text{act}) \cup 2^A \setminus \{\text{act}\}) \]

where \( D(\omega, A, \text{act}) = \{ B \cup \{\text{act}\} | B \in Z(\omega, A \setminus \{\text{act}\}, \text{pre(\text{act})}) \} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \)
take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered
for each action \( \text{act} \) that could cover \( \omega'' \) when fired
ensure that \( \text{act} \) cannot be enabled and take it or throw \( \text{act} \) away.
Characterisation of none-plans, ct’d

Theorem

\[ Z(\omega, A, \omega') = \bigcup_{\omega'' \in \Pi(\omega')} \bigcap_{\text{act} \in A \ \text{eff}(\text{act}) \geq \omega''} (D(\omega, A, \text{act}) \cup 2^A \setminus \{\text{act}\}) \]

where \( D(\omega, A, \text{act}) = \{ B \cup \{\text{act}\} | B \in Z(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act})) \} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \)
take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered
for each action \( \text{act} \) that could cover \( \omega'' \) when fired
ensure that \( \text{act} \) cannot be enabled and take it
or throw \( \text{act} \) away.
Characterisation of none-plans, ct’d

Theorem

\[ \mathcal{Z}(\omega, A, \omega') = \bigcup_{\omega'' \in \Pi(\omega')} \bigcap_{\text{act} \in A_{\text{eff}}(\text{act}) \geq \omega''} (\mathcal{D}(\omega, A, \text{act}) \cup 2^A \setminus \{\text{act}\}) \]

where \( \mathcal{D}(\omega, A, \text{act}) = \{ B \cup \{\text{act}\} \mid B \in \mathcal{Z}(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act})) \} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \)
take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered
for each action \( \text{act} \) that could cover \( \omega'' \) when fired
ensure that \( \text{act} \) cannot be enabled and take it or throw \( \text{act} \) away.
Theorem

\[ Z(\omega, A, \omega') = \bigcup_{\omega'' \in I(\omega')} \bigcap_{\omega \not\geq \omega'' \in I(\omega')} \left( D(\omega, A, \text{act}) \cup 2^A \setminus \{\text{act}\} \right) \]

where \( D(\omega, A, \text{act}) = \{ B \cup \{\text{act}\} \mid B \in Z(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act})) \} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \)
take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered
for each action \( \text{act} \) that could cover \( \omega'' \) when fired
ensure that \( \text{act} \) cannot be enabled and take it
or throw \( \text{act} \) away.
Theorem

\[ \mathcal{Z}(\omega, A, \omega') = \bigcup_{\omega'' \in I(\omega')} \bigcap_{\omega \not\geq \omega'' \atop \text{eff}(act) \geq \omega''} (D(\omega, A, act) \cup 2^A \setminus \{\text{act}\}) \]

where \( D(\omega, A, act) = \{B \cup \{\text{act}\} \mid B \in \mathcal{Z}(\omega, A \setminus \{\text{act}\}, \text{pre}(\text{act}))\} \)

To find all \( B \subseteq A \) that do not make a plan from \( \omega \) to cover \( \omega' \)
take a coordinate \( \omega'' \) of \( \omega' \) that needs to be covered
for each action \( \text{act} \) that could cover \( \omega'' \) when fired
ensure that \( \text{act} \) cannot be enabled and take it
or throw \( \text{act} \) away.
None-plans: tree encoding

\[ \mathcal{Z}((0, 0, 0), \{ \text{makeVehicle, makeCar, makeBoat} \}, (2, 1, 1, 0)) = \bigcup_{\omega \in \mathbb{I}(2,1,1,0)} \mathcal{Z}((0, 0, 0), \{ \text{makeVehicle, makeCar, makeBoat} \}, \omega) = \bigcup_{\omega \in \mathbb{I}(2,1,1,0)} \mathcal{D}((0, 0, 0), \{ \text{makeVehicle, makeCar, makeBoat} \}, \text{makeCar}) \cup 2^A \setminus \{ \text{makeCar} \} \cup \ldots \]
None-plans: tree encoding

\[ Z((0,0,0,0), \{ \text{makeVehicle, makeCar, makeBoat} \}, (2,1,1,0)) = \bigcup_{\omega \in \Pi((2,1,1,0))} Z((0,0,0,0), \{ \text{makeVehicle, makeCar, makeBoat} \}, \omega) = \]

\[ D((0,0,0,0), \{ \text{makeVehicle, makeCar, makeBoat} \}, \text{makeCar}) \cup 2^A \setminus \{ \text{makeCar} \} \cup \ldots \]
None-plans: tree encoding

\[ \mathcal{Z}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, (2, 1, 1, 0)) = \]
\[ \bigcup_{\omega \in \Pi((2, 1, 1, 0))} \mathcal{Z}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, \omega) = \]
\[ \mathcal{D}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, \text{makeCar}) \cup 2^A \setminus \{\text{makeCar}\} \cup \ldots \]
None-plans: tree encoding

\[ \mathcal{Z}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, (2, 1, 1, 0)) = \]
\[ \bigcup_{\omega \in \mathbb{I}((2, 1, 1, 0))} \mathcal{Z}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, \omega) = \]
\[ \mathcal{D}((0, 0, 0, 0), \{\text{makeVehicle, makeCar, makeBoat}\}, \text{makeCar}) \cup 2^A \setminus \{\text{makeCar}\} \cup \ldots \]
None-plans: the full tree unfolding

One can stop unfolding at depth $k$ to underapproximate the none-plan space.
Back to the original domain

SMT-formulae encoding:

- \( \mathcal{AP} \) – encoding of the original domain plan space (courtesy of PlanICS),
- \( \mathcal{CL} \) – blocking sets following from Lemma B,
- \( \mathcal{NOP}^k \) – encoding of the none-plan space unfolding up to \( k \in \mathbb{N} \cup \{\omega\} \)

A new encoding in the original domain plan space:

\[
\widehat{\mathcal{AP}}^k = \mathcal{AP} \land \mathcal{CL} \land \neg \mathcal{NOP}^k
\]

A longer formula: easier or more difficult for an SMT-solver?
Experimental results

Setup:
- random ontologies produced by Ontology Generator
- two experiments/ontology:
  - First – single plan synthesis
  - Total – all plan synthesis

Results for reduction:
- First – usually substantial speedup at some depth
- Total – always substantial speedup at some depth
Experimental results, ct’d

**NoRedTime** – time without reduction

**BestRedTime** – best time with reduction
Conclusions

- A new method for improving efficiency of algorithms solving hard problems,
- A new reduction method for planning,
- Application of the results in the tool PlanICS: quite impressive improvement in some cases,
- SEFM 2015 Best Paper Award.
Thank you!