Generating None-Plans in Order to Find Plans[1]

Wojciech Penczek

a joint work with Michał Knapik and Artur Niewiadomski

Institute of Computer Sciences, PAS, Warsaw, and Siedlce University, Poland

MINI PW, the 24th of November 2016

[1] Best Paper Award at SEFM'15

Outline

Main Contributions

- New web service composition systems - PlanICS and TripICS,
- 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 - Web Service Composition Systems

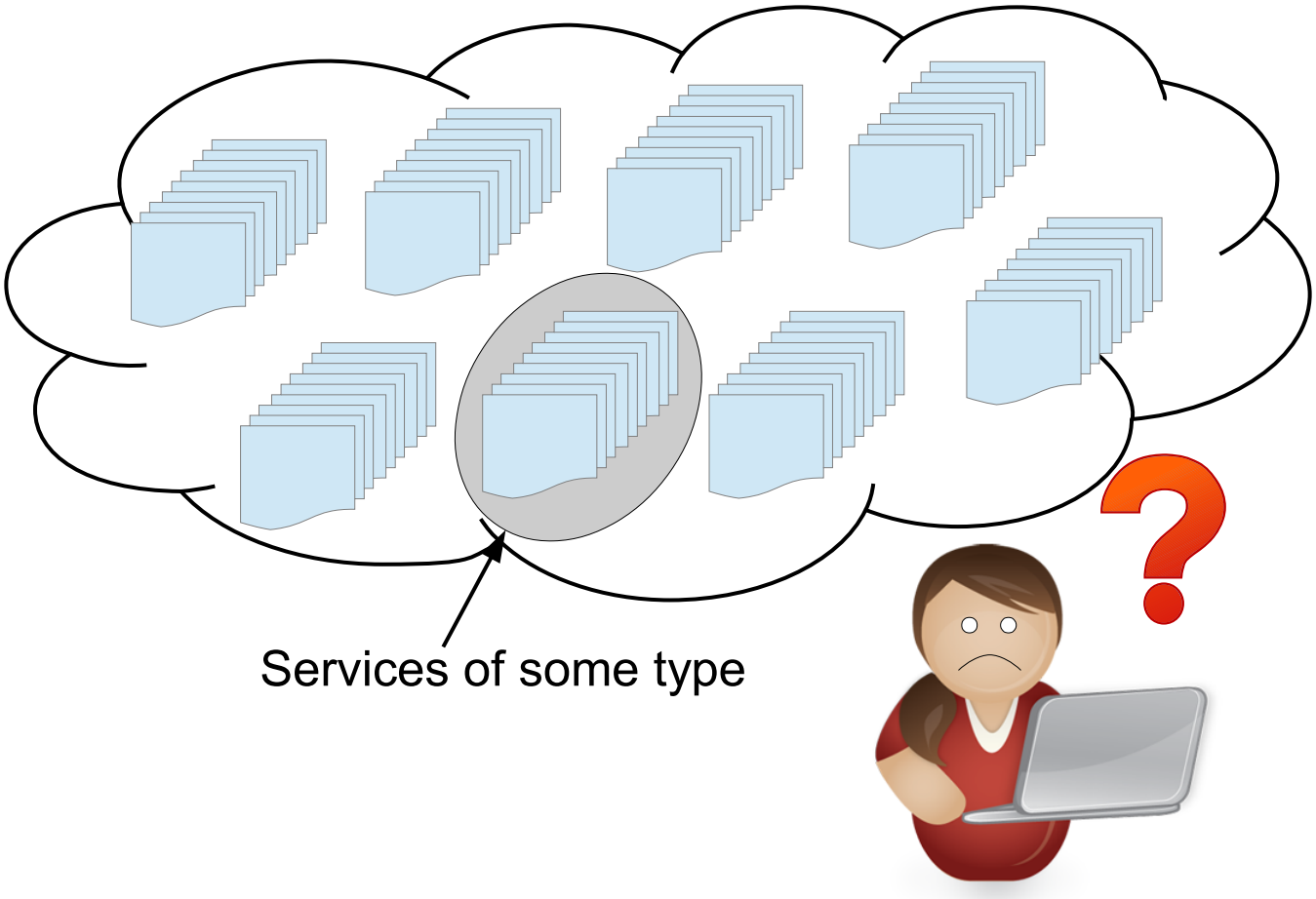
- Entish - IOPR, a two phase planning by an ontology,
- WSMO - ontology, IOPR, a formal goal, embedded rule languages
- WSMX - WSMO implementation, service registration, service discovery by matchmaking, service activation by adapters
- SUPER - composition based on WSMO ontology and AI algorithms
- PlanICS - a state-based approach, a two phase planning, a simple rule language, the abstract planners based on GA and SMT-solvers, the concrete planners based on GA, SA, GEO, and SMT-solvers, and their combinations.

Related Work - Reduction Methods

- Abstraction methods [Cousot, Cousot, ...],
- Partial order reductions [Valmari, Peled, Godefroid, ...],
- Symmetry reductions [Clarke, Emerson, Jha, Sistla, ...],
- CEGAR – Counterexample Guided Abstraction [Clarke et al.],
- and others.

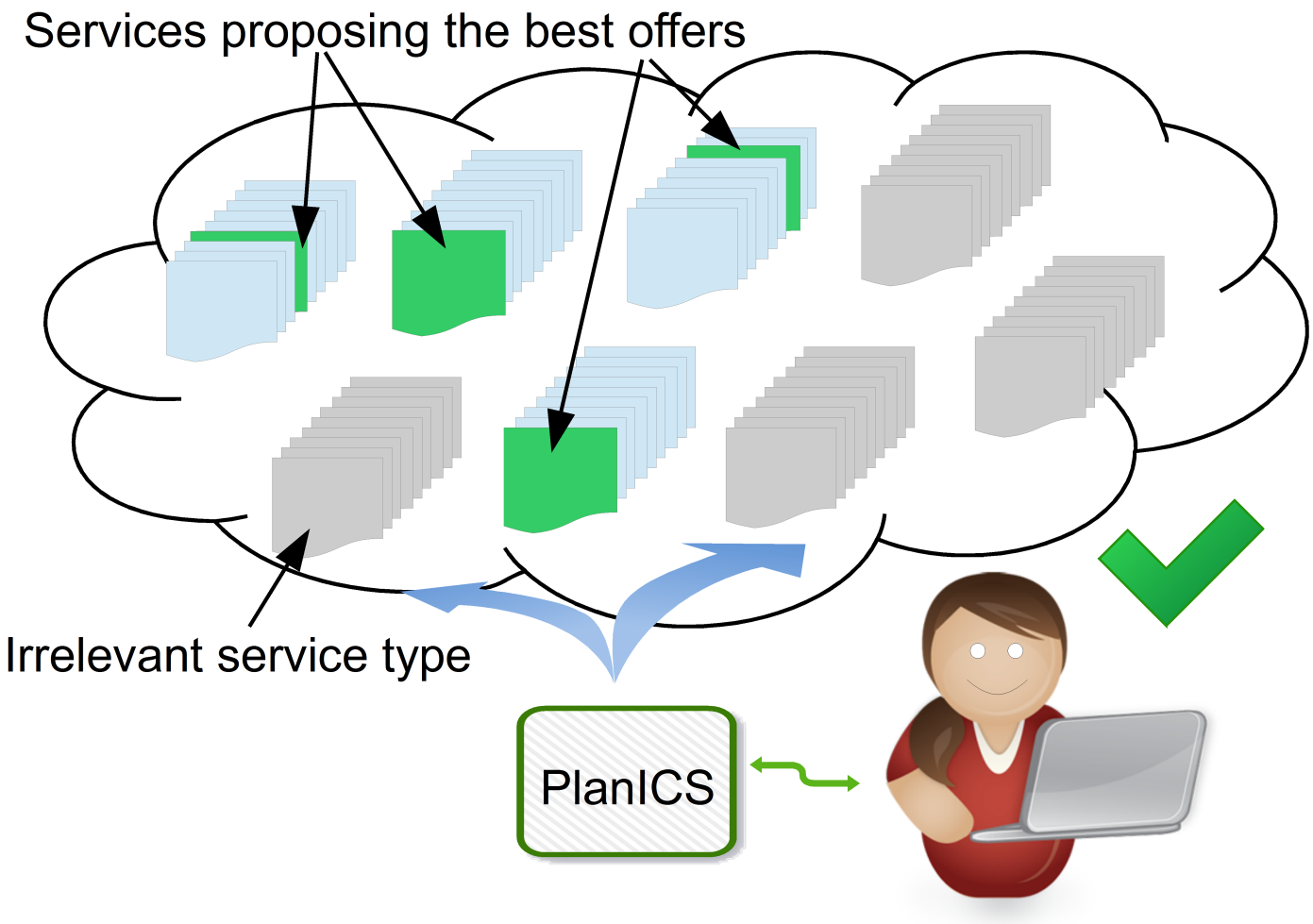
Web Service Composition Problem

Which service types and instances to choose?

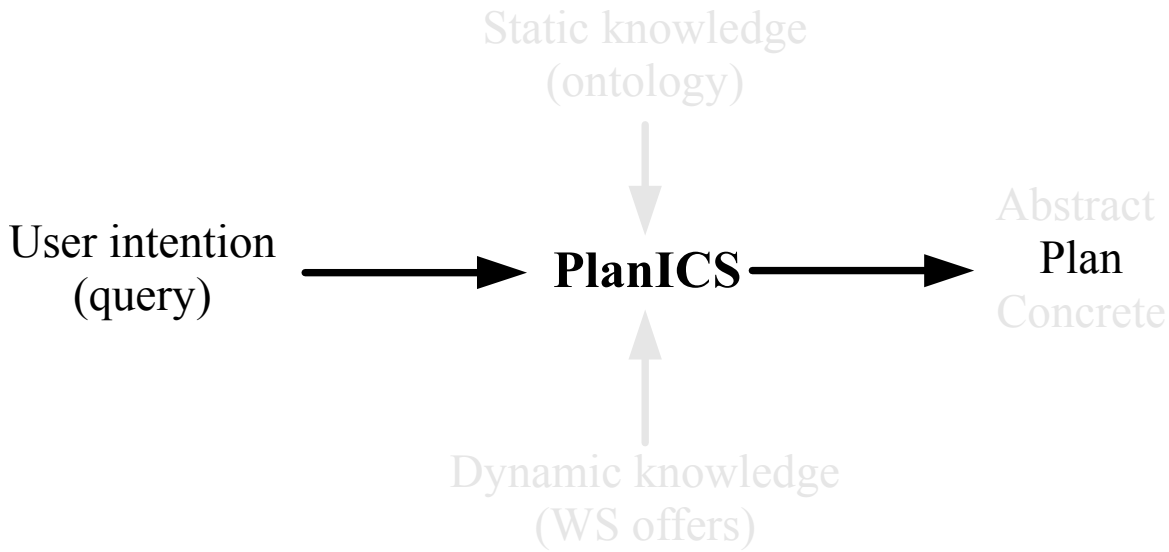


Web Service Composition Problem

PlanICS finds a solution!

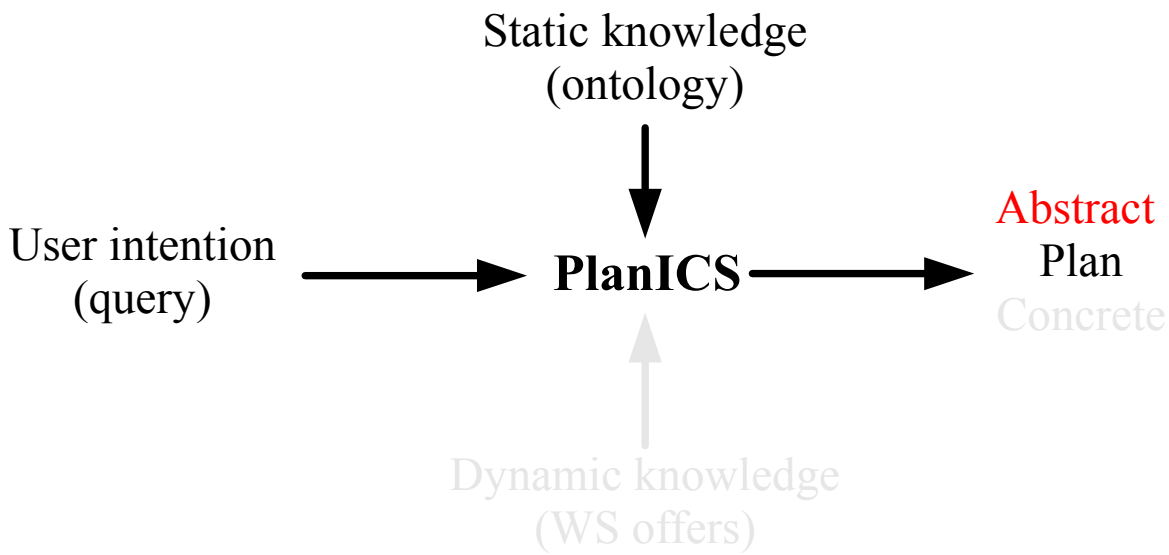


PlanICS - Key Concepts



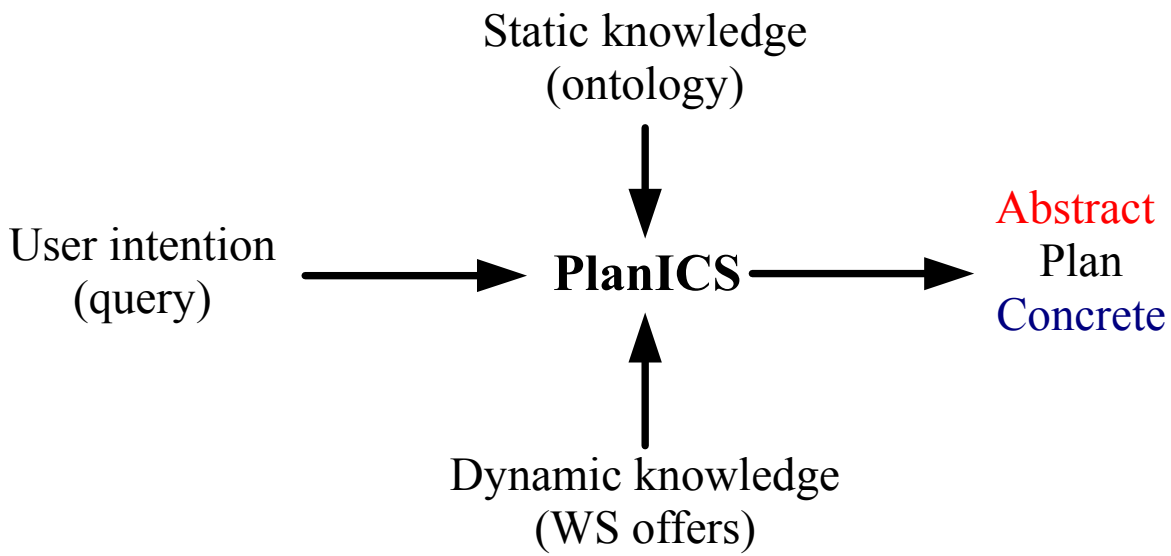
- The main goal: an arrangement of service executions satisfying a user intention
- Ontology - the types of services and objects
- A two phase composition process: abstract (on types) and concrete (on web services)

PlanICS - Key Concepts



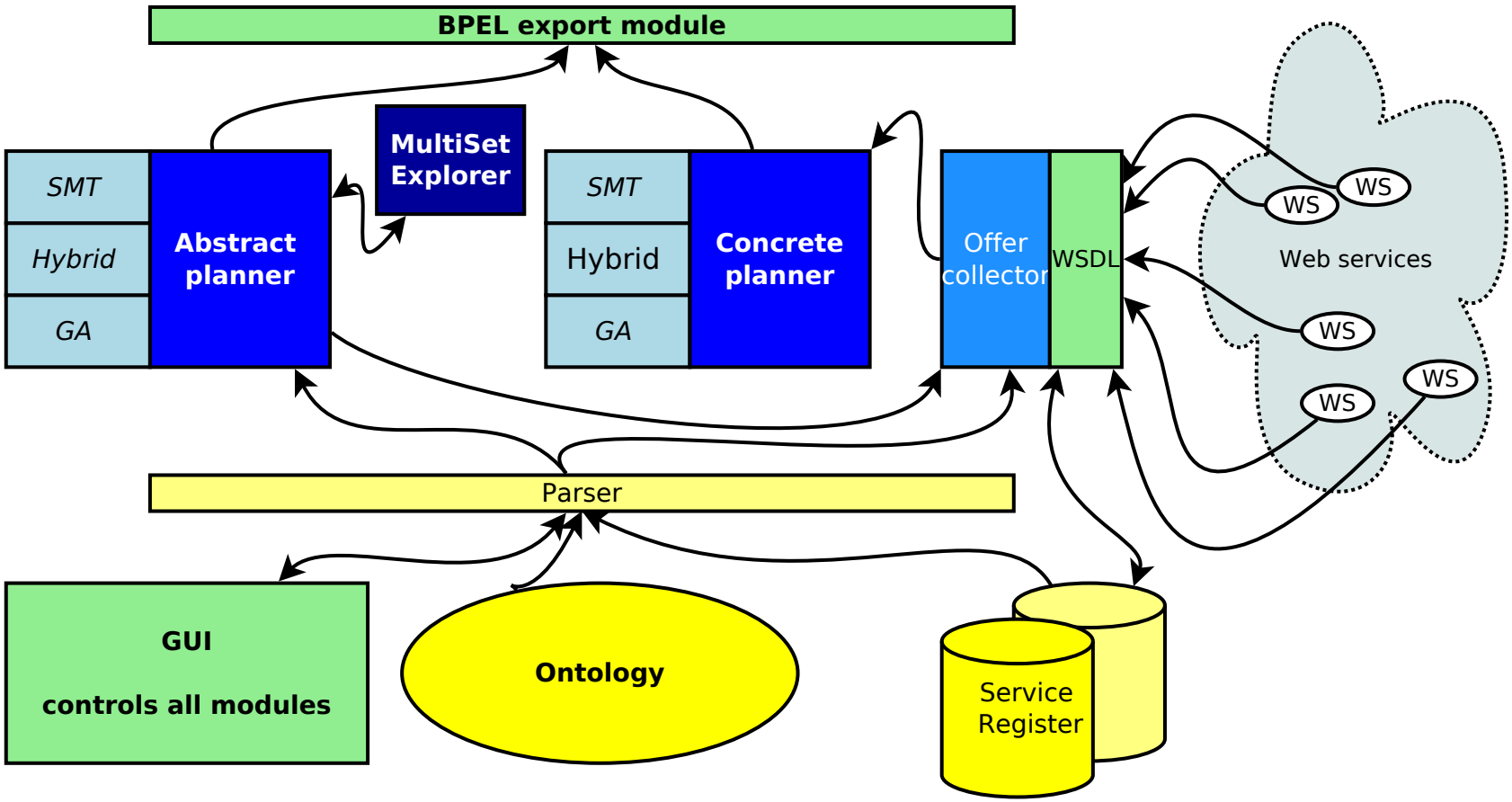
- The main goal: an arrangement of service executions satisfying a user intention
- Ontology - the types of services and objects
- A two phase composition process: abstract (on types) and concrete (on web services)

PlanICS - Key Concepts

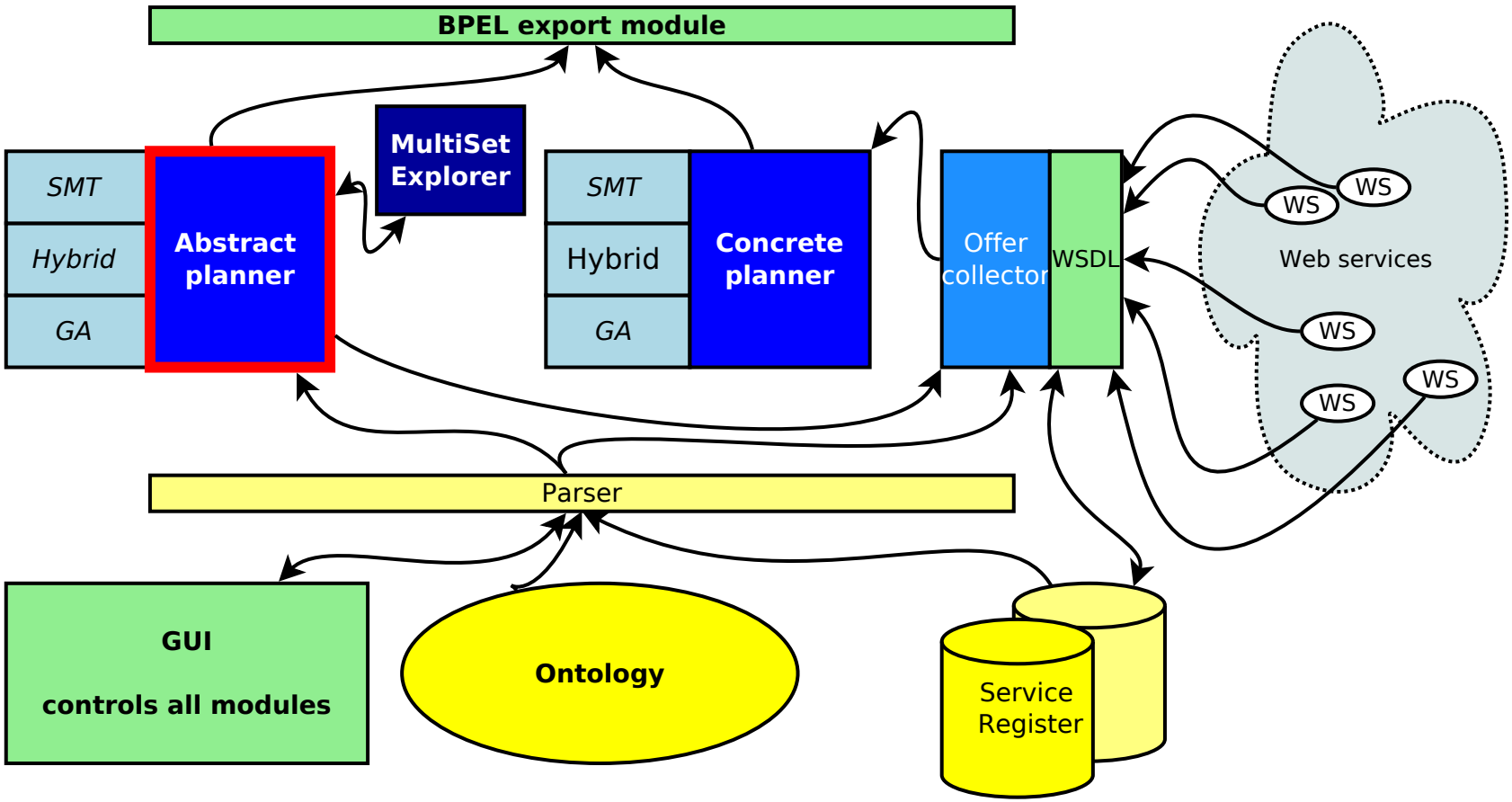


- The main goal: an arrangement of service executions satisfying a user intention
- Ontology - the types of services and objects
- A two phase composition process: abstract (on types) and concrete (on web services)

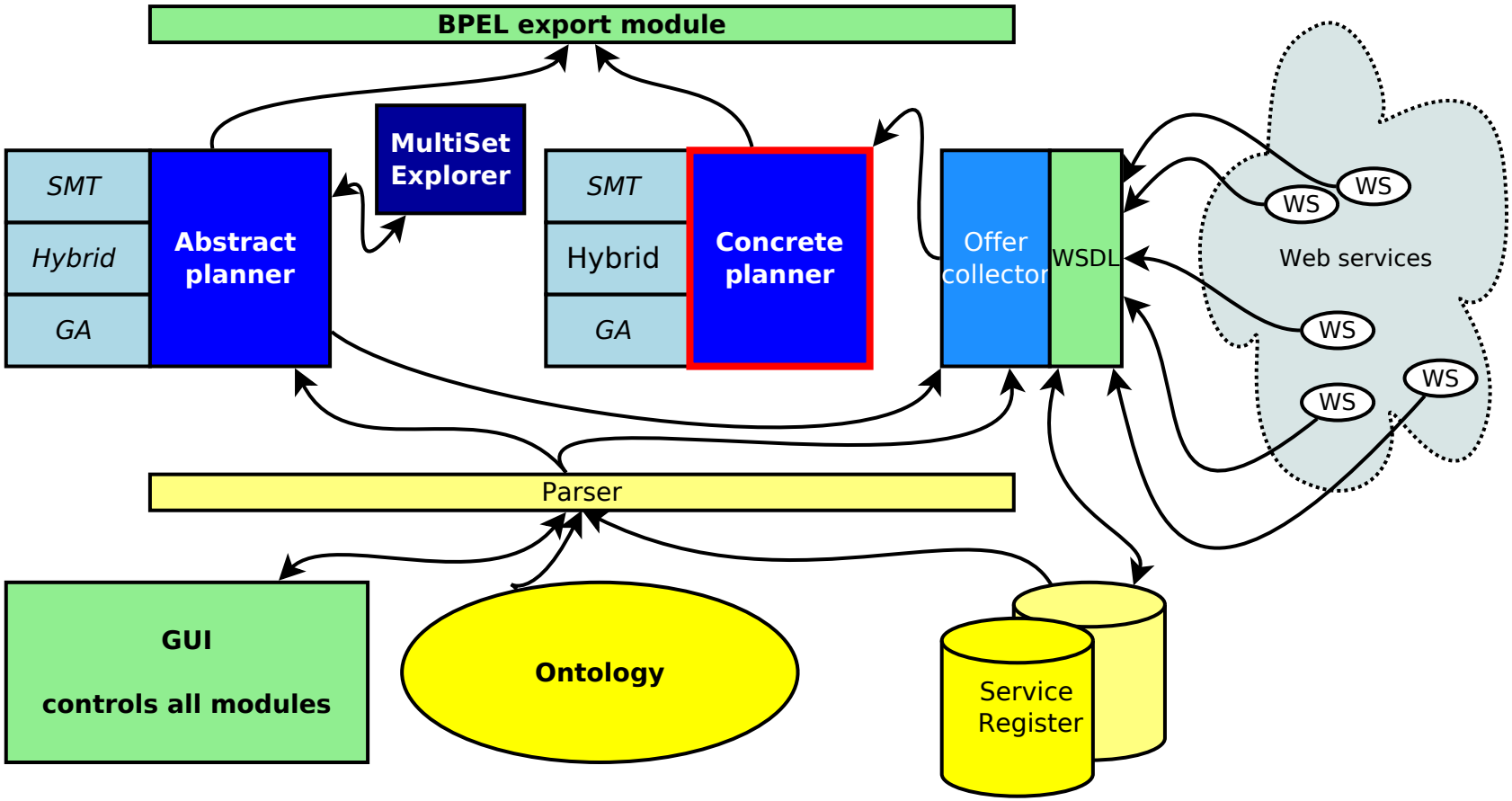
Architecture of PlanICS



Architecture of PlanICS



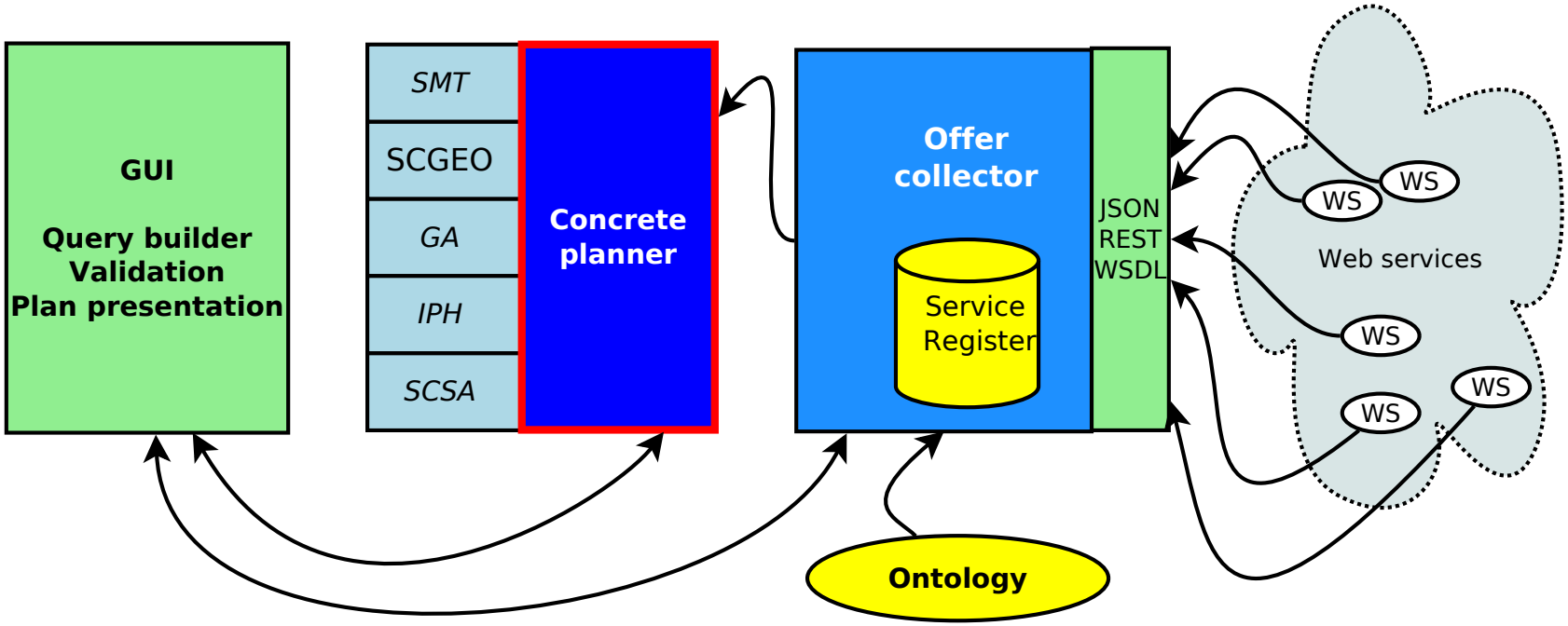
Architecture of PlanICS



TripICS - Main Ideas

- TripICS - a real-life application of PlanICS to planning trips and travels around the world,
- TripICS - a specialization of the concrete planning viewed as a constrained optimization problem to the ontology containing services provided by hotels, airlines, railways, museums etc.,
- TripICS finds an optimal plan satisfying the user's requirements by applying the concrete planners of PlanICS,
- Abstract planning of TripICS - semi-automatic, the user builds an abstract plan using a Graphical User Interface.

Architecture of TripICS



General System Description

TripICS

A user-friendly planning of visits to cities (places), travels, and entertainments, satisfying the user's requirements.

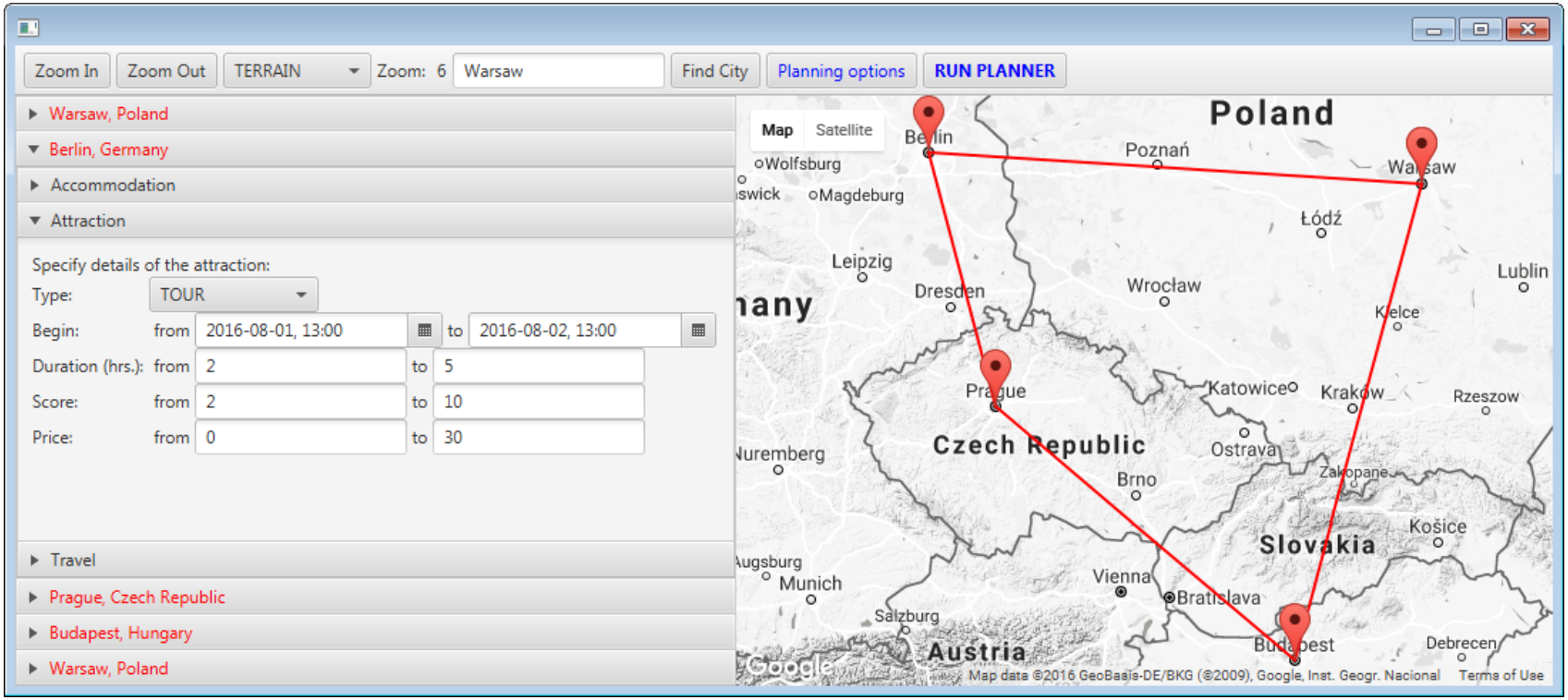
User's Requirements

Specification of cities, places, dates, means of travels, and quality requirements (level of prices, quality of hotels, etc.).

Optimal Plan

A plan of highest quality according to the given requirements.

Specifying Requirements



Plan Returned by TripICS

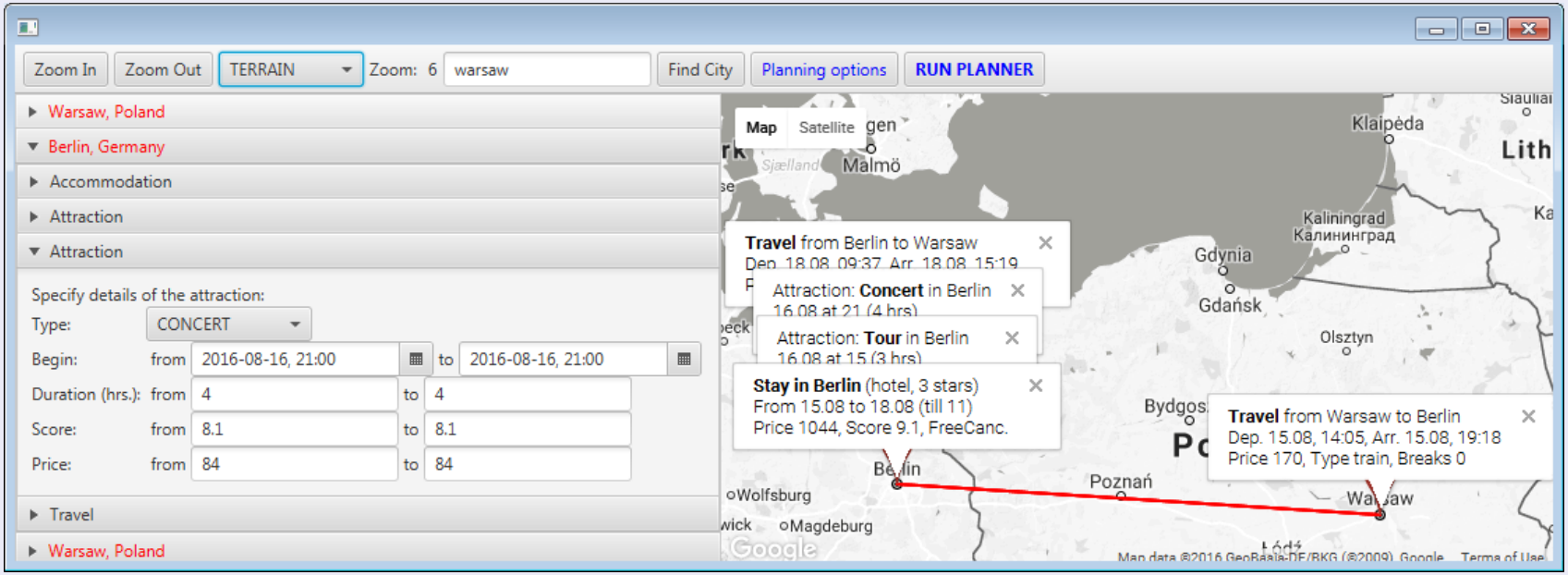


Figure: The example plan

General idea – intuition

- D – a domain to search for a **plan** (difficult task!),

- D' – an abstract domain in which finding a **plan** is easier,
- 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',

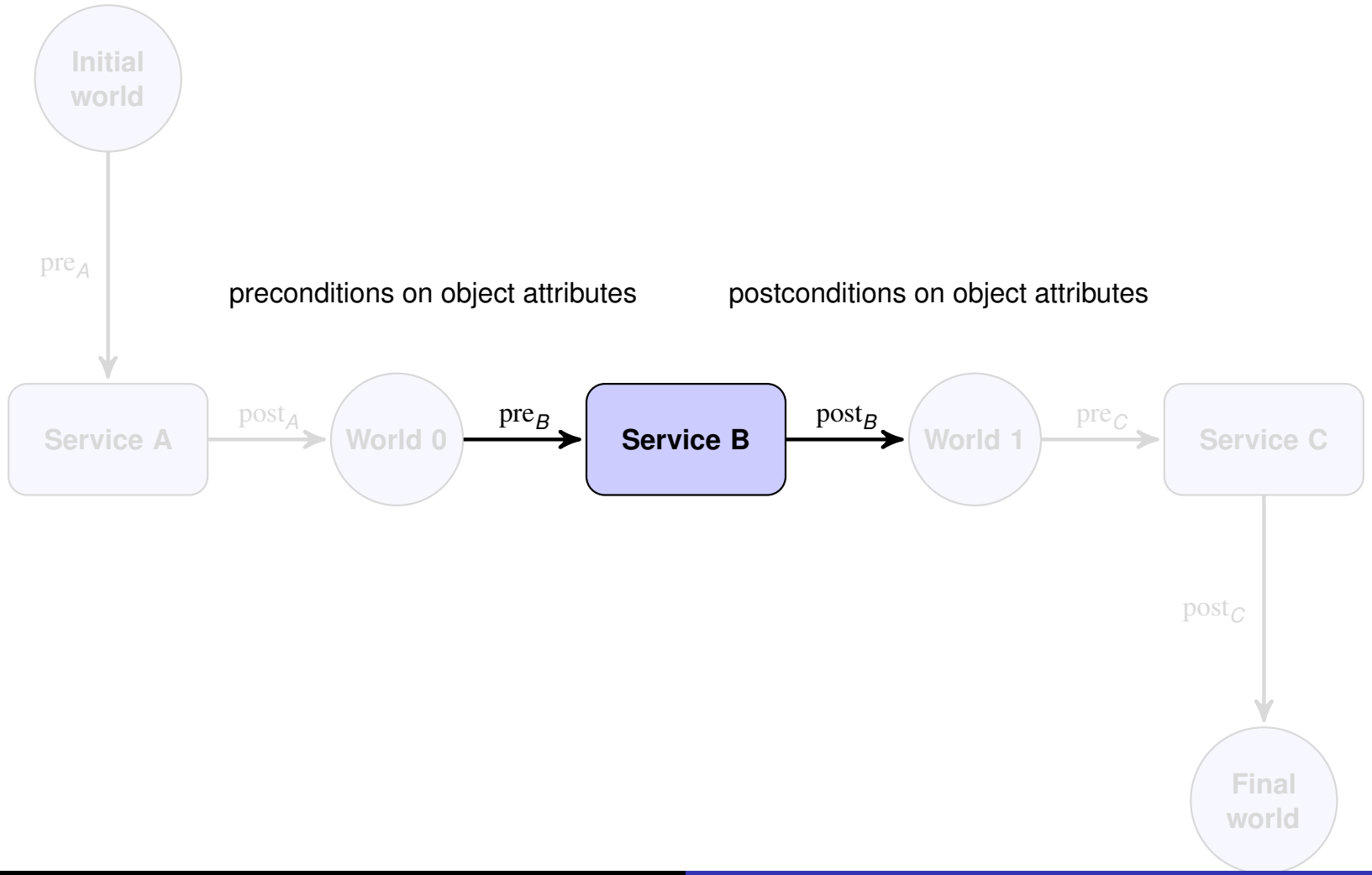
- prune D from (the) **none-plans** of D',
- search for (the) **plans** in D pruned.

Application to planning in PlanICS

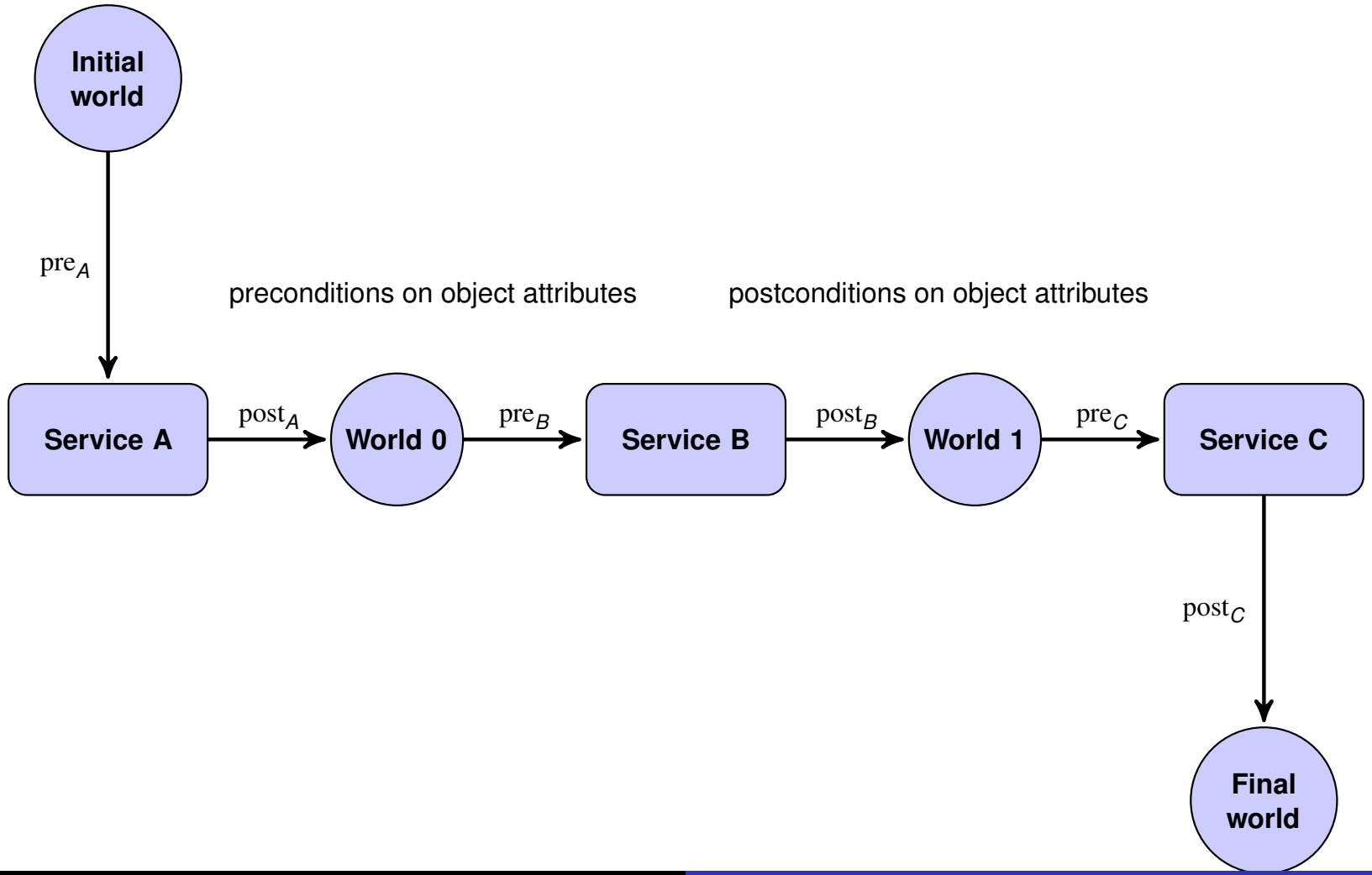
- Given an ontology of object types and services (OWL-like language),
- Given a user query: (initial worlds, final worlds),
- A **world** – a set of objects (each object has a type and attributes),
- A **service**: **(in, inout, out, pre, post)**, where in, inout, out are sets of objects,
- **pre** – a boolean formula over the object attributes of **in** and **inout**,
- **post** – a boolean formula over the object attributes of **inout** and **out**.

Task: Find all plans from some initial to some final world.
This problem is NP-complete.

Service composition in PlanICS



Service composition in PlanICS



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** – **(precondition, effect)**,
- **precondition** – a multiset of objects (objects required),
- **effect** – a multiset of objects (new objects added).

- Let B be a set of **services**,
- B' – the set of **simplified services** of B,
- **The main Property of Abstraction:** If B' **cannot** be composed into a **plan**, then B **cannot either**.
- **Goal:** synthesize constraints of non-composability.

(Abstract) Planning Domain

(Abstract) 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 $\mathrm{act} \in Act$:

- $\mathrm{pre}(\mathrm{act})$ – **precondition** of act,
- $\mathrm{eff}(\mathrm{act})$ – **effect** of act.

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

Action $\mathrm{act} \in Act$ is **enabled** in $\omega \in \mathcal{W}_{\mathcal{H}}$ iff $\mathrm{pre}(\mathrm{act}) \leq \omega$ and the results of firing act: $\omega \xrightarrow{\mathrm{act}} \omega + \mathrm{eff}(\mathrm{act})$

Plans

Given $\mathcal{P} = (\mathcal{W}_{\mathcal{H}}, F_I, F_G, Act)$, $\mathbf{B} \subseteq Act$

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

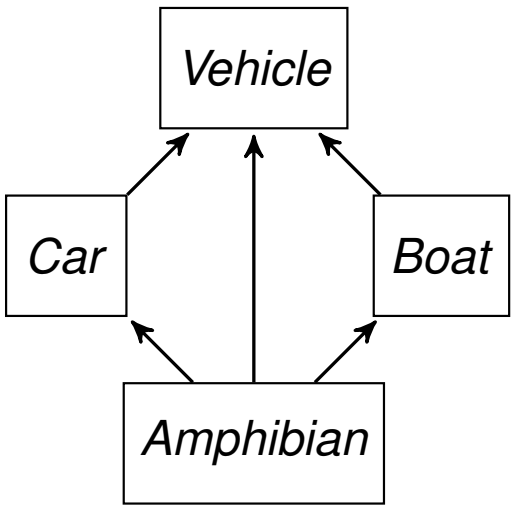
$$\pi = \omega_0 \overset{\mathrm{act}_1}{\rightarrow} \omega_1 \overset{\mathrm{act}_2}{\rightarrow} \ldots \overset{\mathrm{act}_{n-1}}{\rightarrow} \omega_{n-1} \overset{\mathrm{act}_n}{\rightarrow} \omega_n$$

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

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

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

Exemplary planning domain

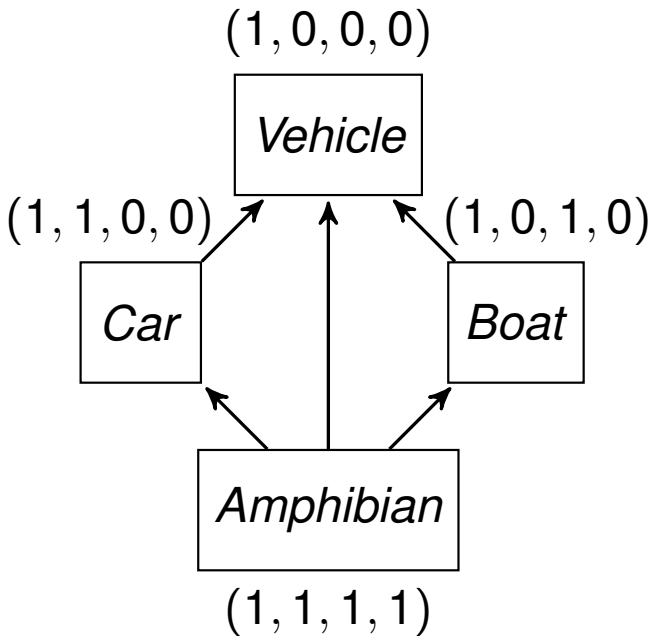


Vehicles' inheritance

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



Vehicles' inheritance

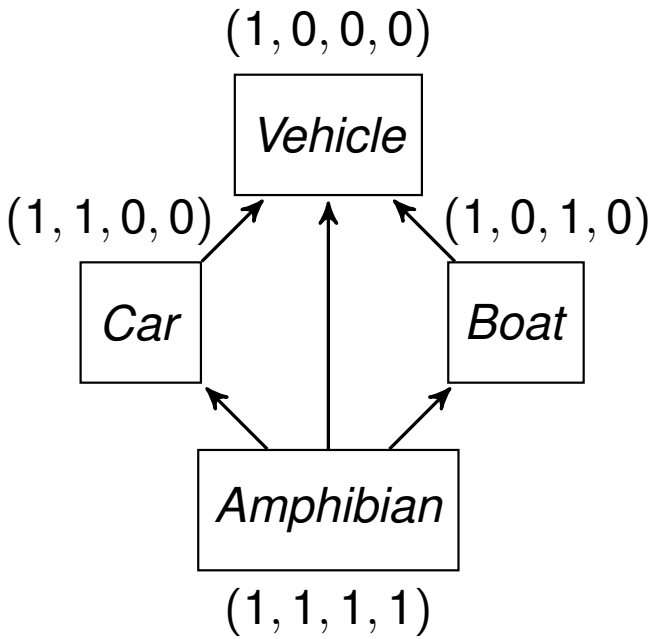
Order of the objects:
(*Vehicle*, *Car*, *Boat*, *Amphibian*)

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

  $\mathrm{pre}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{0}) + (\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0}) = (\mathbf{2},\mathbf{1},\mathbf{1},\mathbf{0})$

  $\mathrm{eff}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$

Exemplary planning domain, ct'd



Vehicles' inheritance

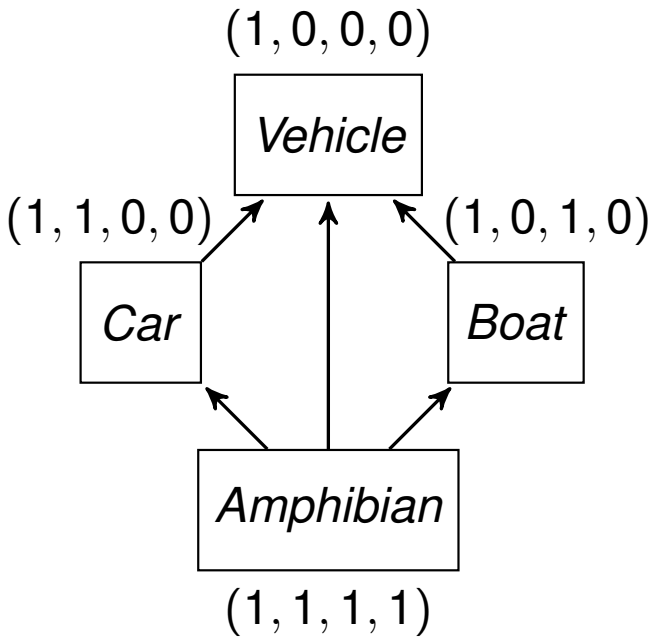
Order of the objects:
(*Vehicle*, *Car*, *Boat*, *Amphibian*)

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

  $\mathrm{pre}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{0}) + (\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0}) = (\mathbf{2},\mathbf{1},\mathbf{1},\mathbf{0})$

  $\mathrm{eff}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$

Exemplary planning domain, ct'd



Vehicles' inheritance

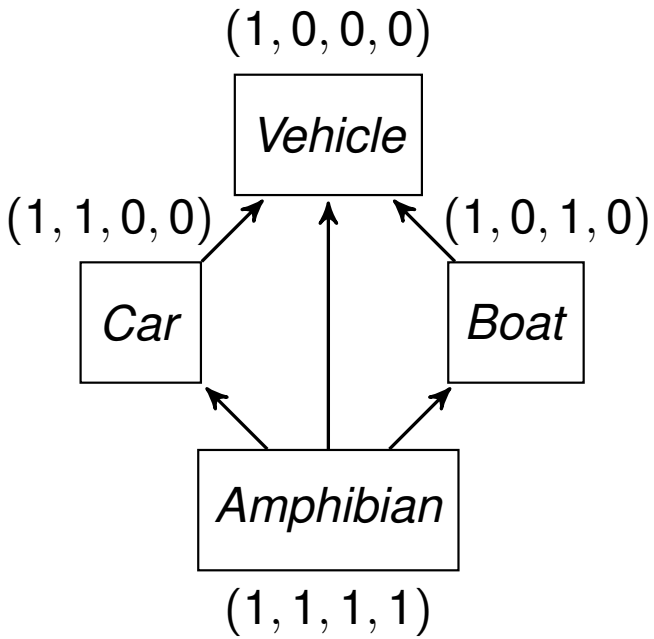
Order of the objects:
(*Vehicle*, *Car*, *Boat*, *Amphibian*)

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

  $\mathrm{pre}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{0}) + (\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0}) = (\mathbf{2},\mathbf{1},\mathbf{1},\mathbf{0})$

  $\mathrm{eff}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$

Exemplary planning domain, ct'd



Vehicles' inheritance

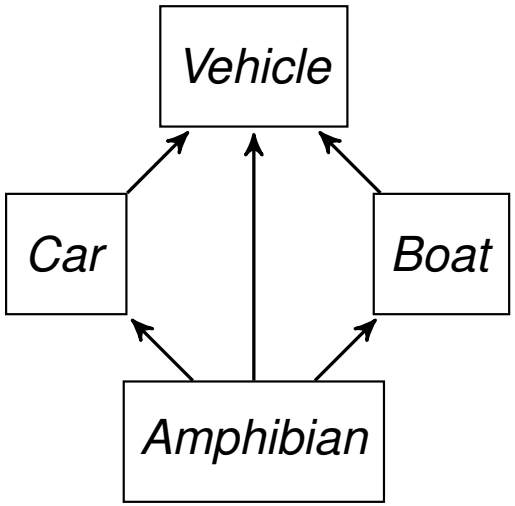
Order of the objects:
(*Vehicle*, *Car*, *Boat*, *Amphibian*)

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

  $\mathrm{pre}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{0},\mathbf{1},\mathbf{0}) + (\mathbf{1},\mathbf{1},\mathbf{0},\mathbf{0}) = (\mathbf{2},\mathbf{1},\mathbf{1},\mathbf{0})$

  $\mathrm{eff}(\textit{makeAmphibian}) = (\mathbf{1},\mathbf{1},\mathbf{1},\mathbf{1})$

Exemplary planning domain, ct'd



Vehicles' inheritance

Actions:

- $\mathrm{pre}(\mathit{makeVehicle}) = (0,0,0,0)$
  $\mathrm{eff}(\mathit{makeVehicle}) = (1,0,0,0)$
- $\mathrm{pre}(\mathit{makeCar}) = (1,0,0,0)$
  $\mathrm{eff}(\mathit{makeCar}) = (1,1,0,0)$
- $\mathrm{pre}(\mathit{makeBoat}) = (1,0,0,0)$
  $\mathrm{eff}(\mathit{makeBoat}) = (1,0,1,0)$
- $\mathrm{pre}(\mathit{makeAmphibian}) = (2,1,1,0)$
  $\mathrm{eff}(\mathit{makeAmphibian}) = (1,1,1,1)$
- $\mathrm{pre}(\mathit{tinker}) = (2,2,1,1)$
  $\mathrm{eff}(\mathit{tinker}) = (2,2,2,2)$

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

Classifying actions

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

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

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

$\omega \in \mathcal{W}_{\mathcal{H}}$, $i > 0$

- $G_0^\omega = \{\text{act} \in Act \mid pre(\text{act}) \leq \omega\}$ – the actions enabled in ω,
- $G_{i+1}^\omega = \text{enact}(G_i^\omega)$ – the actions enabled in i–th step
- $H_0^\omega = G_0^\omega$,
- $H_{i+1}^\omega = G_{i+1}^\omega \setminus G_i^\omega$ – the actions **newly** enabled in i–th step.

Classifying actions, ct'd

$\omega, \omega' \in \mathcal{W}_{\mathcal{H}}$

$$\mathrm{kgoal}(\omega, \omega') = \mathit{min}(\{k \in \mathbb{N} \mid \sum_{\mathrm{act} \in G_k^{\omega}} V_{\max} \cdot \mathrm{eff}(\mathrm{act}) \geq \omega'\})$$

the minimal step at which greedily fired actions cover ω'.

Lemma A

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

Planning in $\mathcal{P}$ is easy.

Classifying actions

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

First easy reductions:

- block all sets of actions that do not satisfy Lemma B.

More reductions: consider none-plans.

None-plans

$\mathbf{A} \subseteq \mathit{Act}, \omega, \omega' \in \mathcal{W}_{\mathcal{H}}$

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

None-plan: a set of actions B which is not a support of any plan starting at ω and covering ω'.

$\mathbb{I}(\omega) := \{\omega' \mid \|\omega'\| = 1 \wedge \omega \geq \omega'\}$ – unitary coordination vectors of ω

e.g., $\mathbb{I}((2,1,1,0)) = \{(1,0,0,0),(0,1,0,0),(0,0,1,0)\}$

Characterisation of none-plans

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{act \in A \\ \text{eff}(act) \geq \omega''}} \left(\mathcal{D}(\omega, A, act) \cup 2^{A \setminus \{act\}}\right)$$

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

Characterisation of none-plans, ct'd

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\mathrm{act} \in A \\ \mathrm{eff}(\mathrm{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \mathrm{act}) \cup 2^{A \setminus \{\mathrm{act}\}} \right)$$

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

To find all $B \subseteq A$ that do not make a plan from ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

Characterisation of none-plans, ct'd

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\text{act} \in A \\ \text{eff}(\text{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \text{act}) \cup 2^{A \setminus \{\text{act}\}} \right)$$

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 ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

Characterisation of none-plans, ct'd

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\text{act} \in A \\ \text{eff}(\text{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \text{act}) \cup 2^{A \setminus \{\text{act}\}} \right)$$

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 ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

Characterisation of none-plans, ct'd

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\mathrm{act} \in A \\ \mathrm{eff}(\mathrm{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \mathrm{act}) \cup 2^{A \setminus \{\mathrm{act}\}} \right)$$

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

To find all $B \subseteq A$ that do not make a plan from ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

Characterisation of none-plans, ct'd

Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\mathrm{act} \in A \\ \mathrm{eff}(\mathrm{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \mathrm{act}) \cup 2^{A \setminus \{\mathrm{act}\}} \right)$$

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

To find all $B \subseteq A$ that do not make a plan from ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

Characterisation of none-plans, ct'd

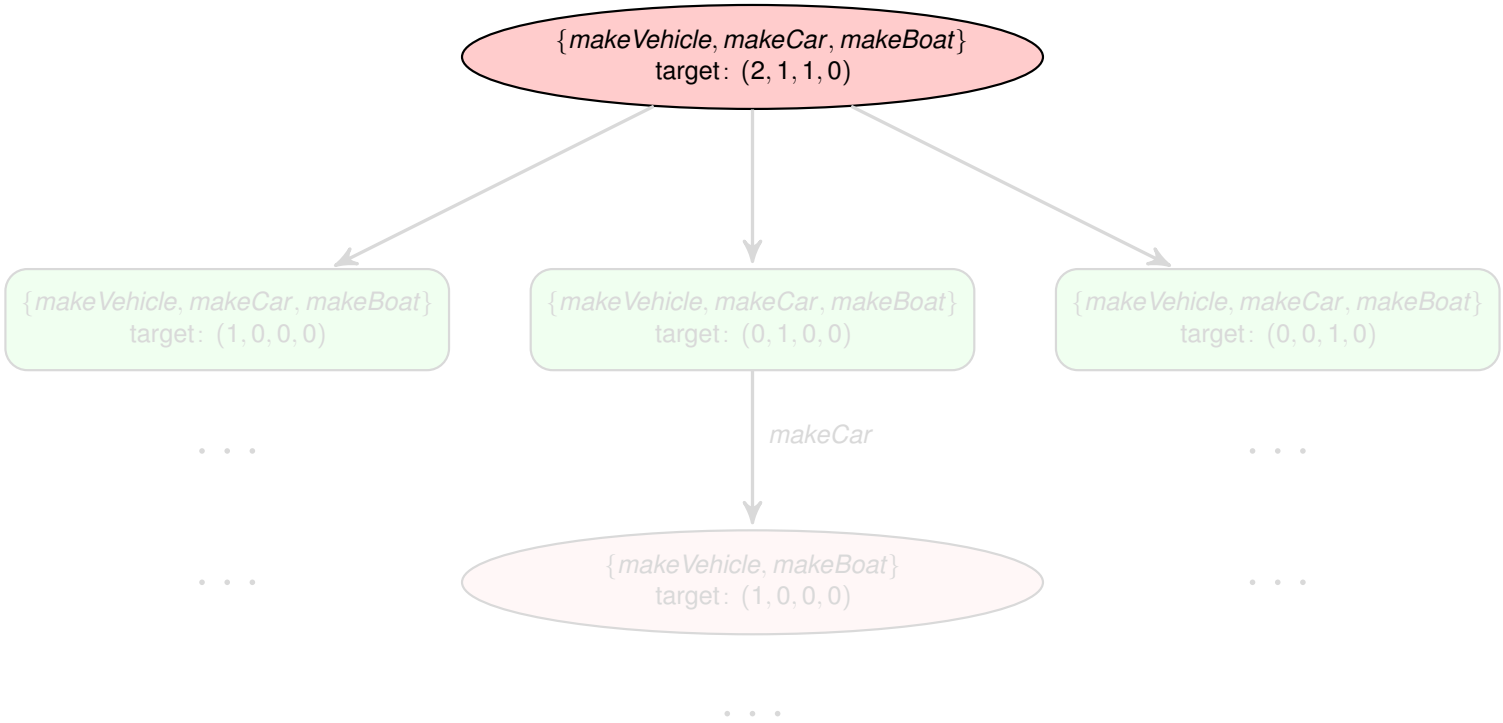
Theorem

$$\mathcal{Z}(\omega, A, \omega') = \bigcup_{\substack{\omega'' \in \mathbb{I}(\omega') \\ \omega \not\geq \omega''}} \bigcap_{\substack{\mathrm{act} \in A \\ \mathrm{eff}(\mathrm{act}) \geq \omega''}} \left(\mathcal{D}(\omega, A, \mathrm{act}) \cup 2^{A \setminus \{\mathrm{act}\}} \right)$$

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

To find all $B \subseteq A$ that do not make a plan from ω to cover ω'
take a coordinate ω'' of ω' that needs to be covered
for each action act that could cover ω'' when fired
ensure that either act cannot be enabled and take it
or throw act away.

None-plans: tree encoding

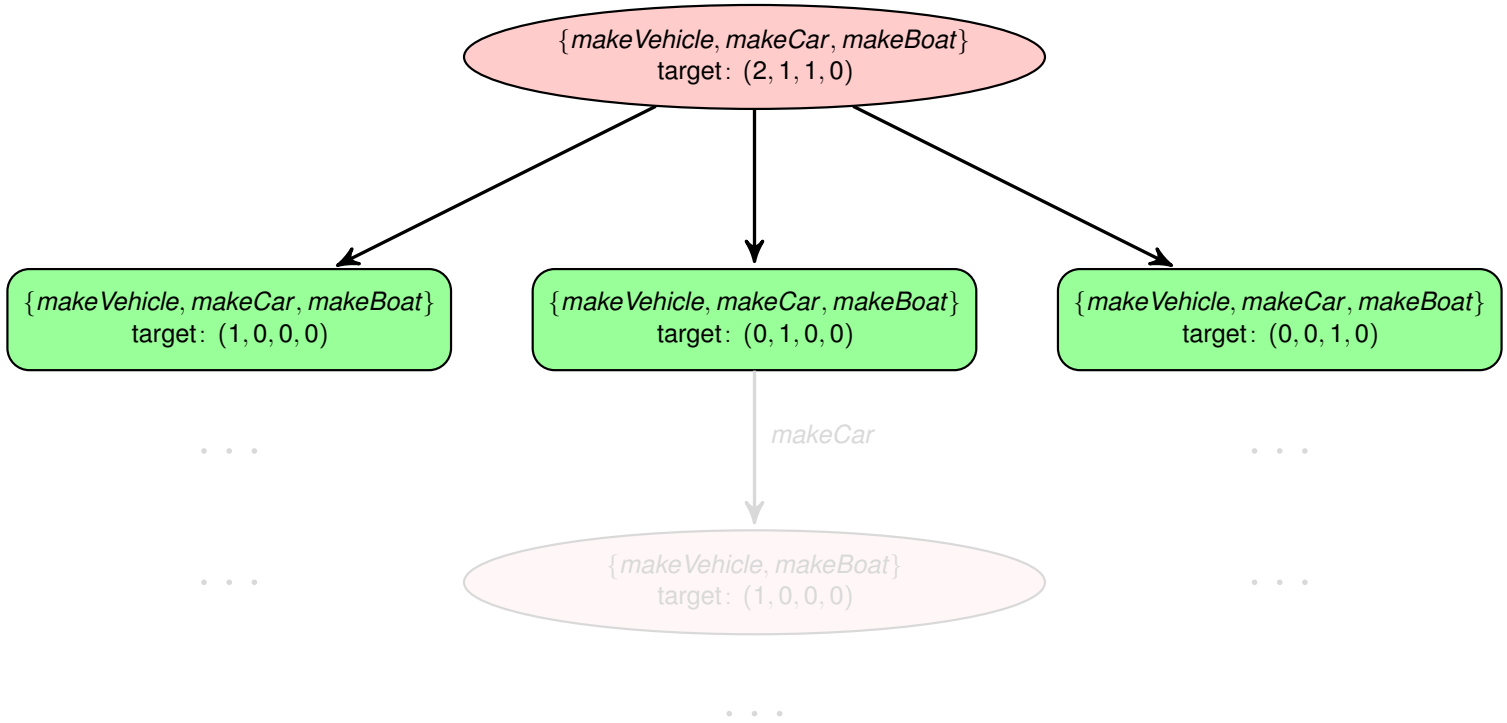


$$\mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},(2,1,1,0)) =$$

$$\bigcup_{\omega \in \mathbb{I}((2,1,1,0))} \mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},\omega) =$$

$$\mathcal{D}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\}, makeCar) \cup 2^{A \setminus \{makeCar\}} \cup \ldots$$

None-plans: tree encoding

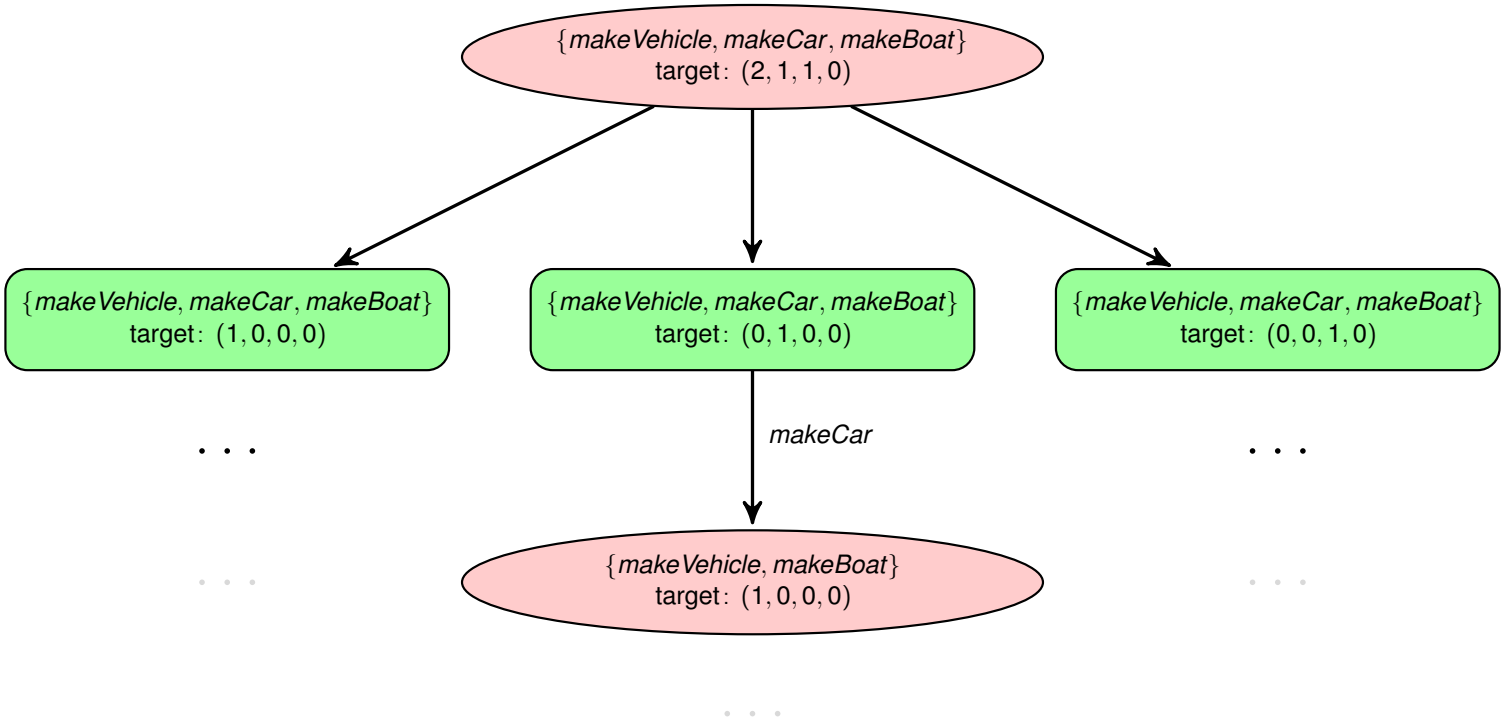


$$\mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},(2,1,1,0)) =$$

$$\bigcup_{\omega \in \mathbb{I}((2,1,1,0))} \mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},\omega) =$$

$$\mathcal{D}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\}, makeCar) \cup 2^{A \setminus \{makeCar\}} \cup \ldots$$

None-plans: tree encoding

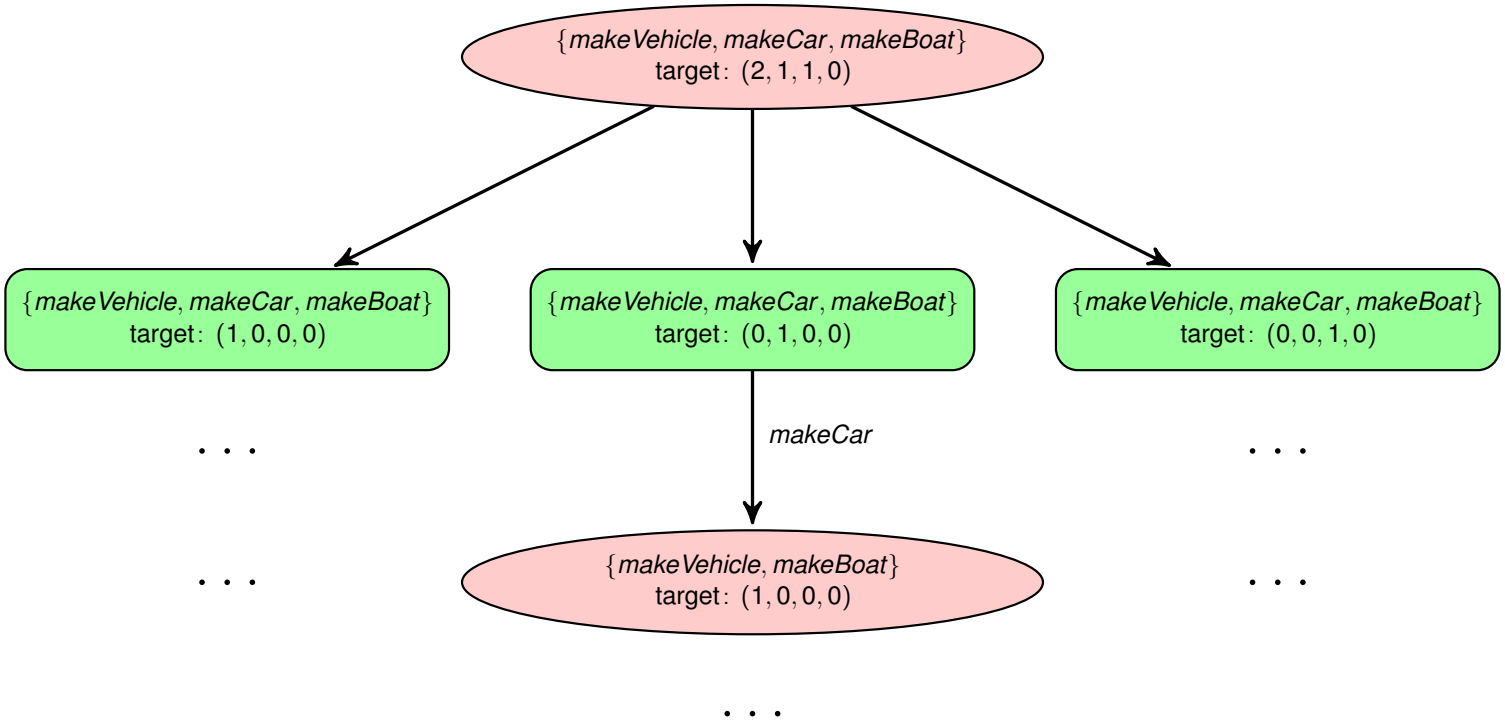


$$\mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},(2,1,1,0)) =$$

$$\bigcup_{\omega \in \mathbb{I}((2,1,1,0))} \mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},\omega) =$$

$$\mathcal{D}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\}, makeCar) \cup 2^{A \setminus \{makeCar\}} \cup \ldots$$

None-plans: tree encoding

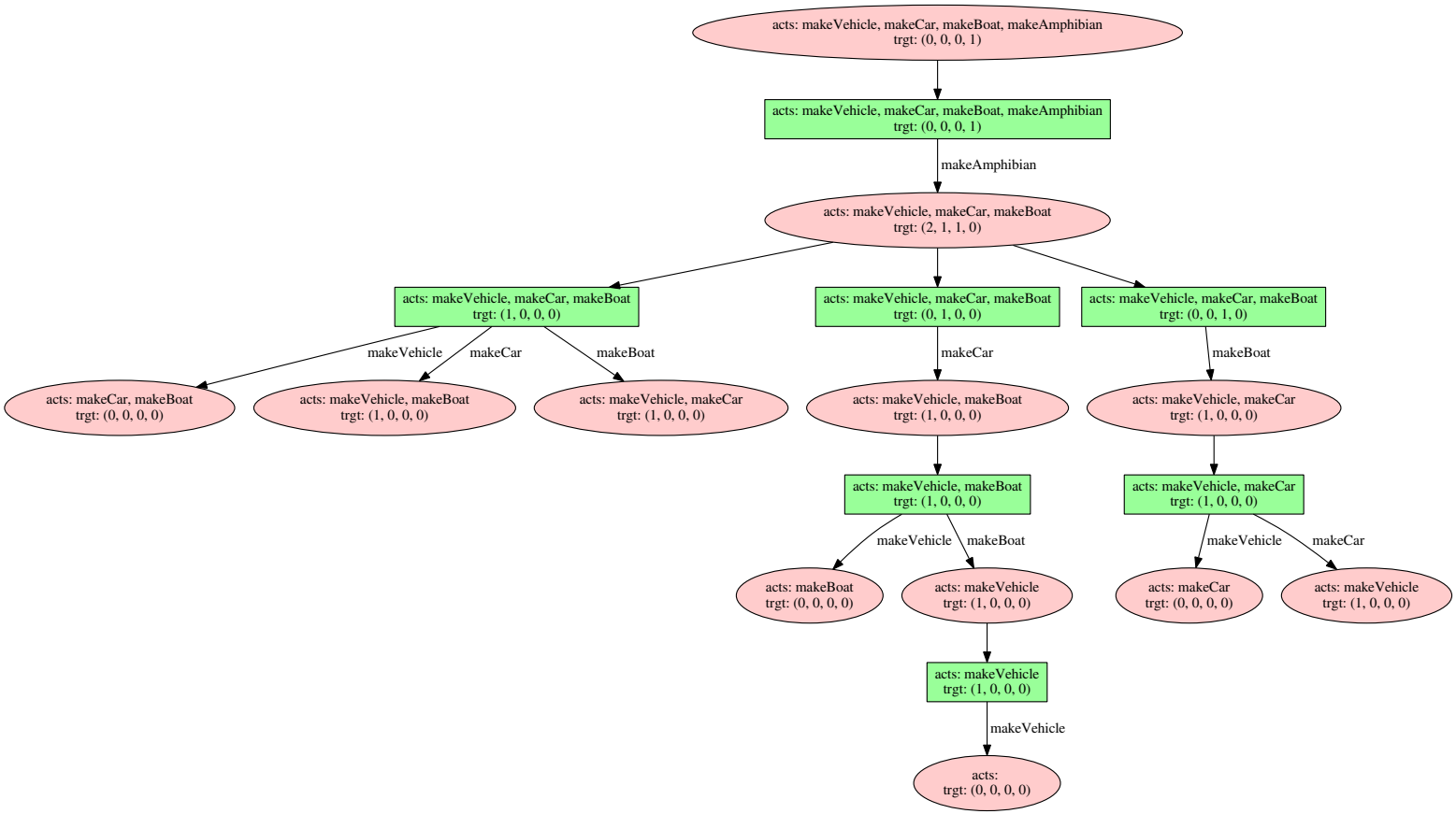


$$\mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},(2,1,1,0)) =$$

$$\bigcup_{\omega \in \mathbb{I}((2,1,1,0))} \mathcal{Z}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\},\omega) =$$

$$\mathcal{D}((0,0,0,0),\{makeVehicle, makeCar, makeBoat\}, makeCar) \cup 2^{A \setminus \{makeCar\}} \cup \ldots$$

None-plans: the full tree unfolding



One can stop unfolding at depth k to underapproximate the none-plan space.

Back to the original domain

The SMT-formulae:

- $\mathcal{AP}$ – 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:

$$\widetilde{\mathcal{AP}}^k = \mathcal{AP} \wedge \mathcal{CL} \wedge \neg \mathcal{NOP}^k$$

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

Experimental results

Setup:

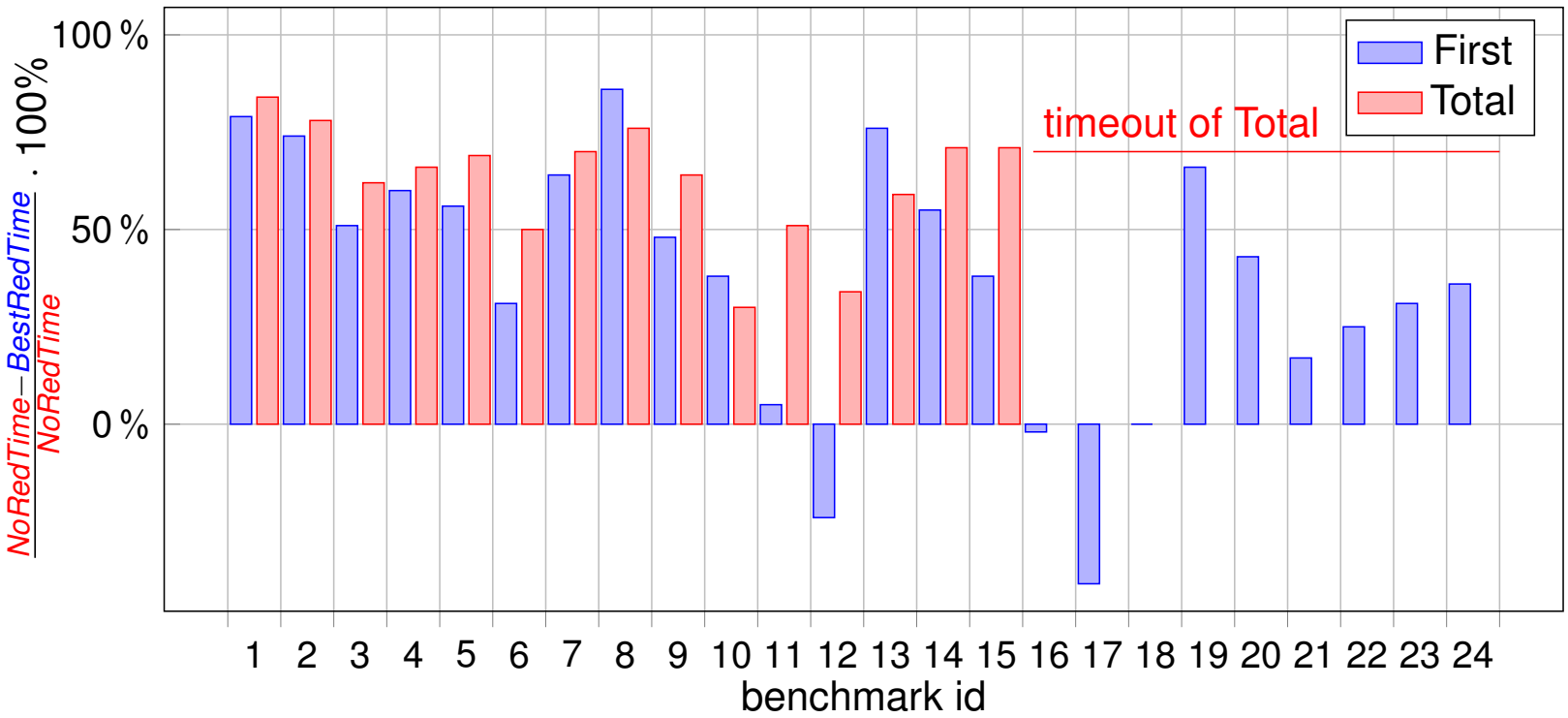
- random ontologies produced by Ontology Generator
- two experiments/ontology:

  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 web composition system PlanICS - http://kus.ii.uph.edu.pl/en/
- 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.

Thank you!