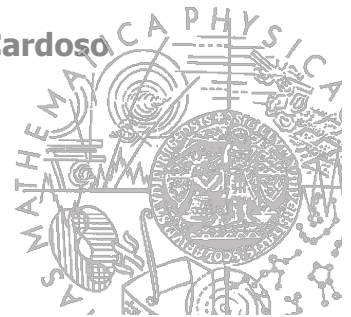


Attribute Grammars for Modeling Planning Domains

Roman Barták

Charles University, Czech Republic

with contributions from **Adrien Maillard**, **Rafael C. Cardoso**



Model vs. model-free approaches

Model-free approaches

- easy to use (+)
- good results (+)
- black box (-)
- „strange“ mistakes (-)

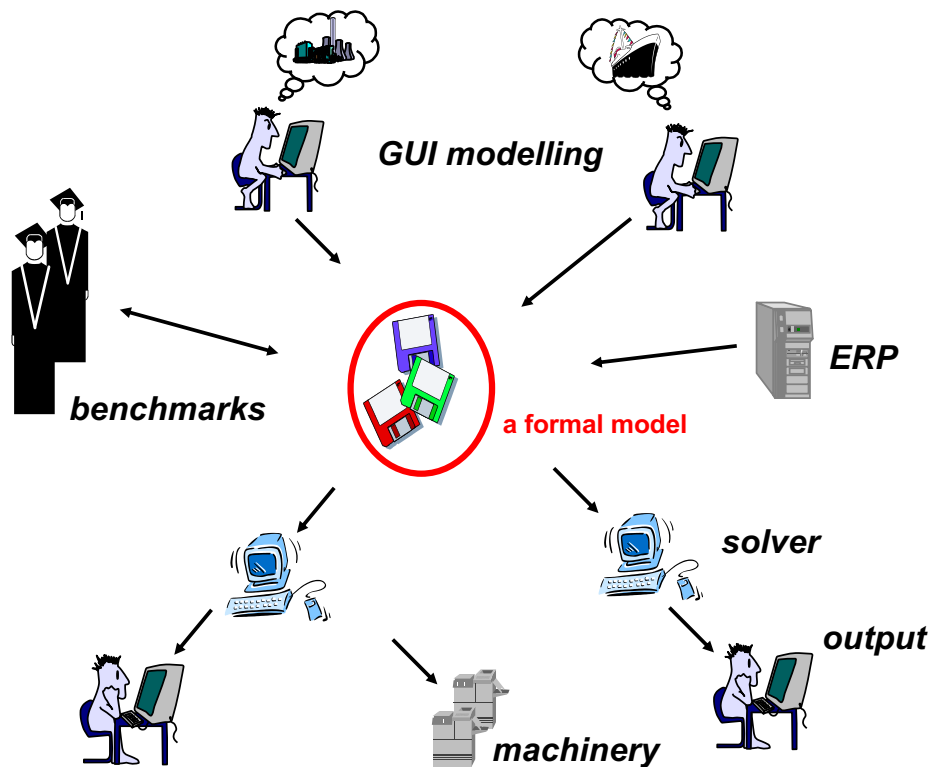


Indian elephant

Model-based approaches

- more complex design of models (-)
- explanation of results (+)
- model verification (+)

A model centric approach [KEPS 2003]



Attribute grammars

a **context-free** (CF) grammar, where the symbols are annotated by sets of **attributes** that can be connected via **constraints**

$a^i b^j c^k$ (CF grammar)

$S \rightarrow A.B.C$

$A \rightarrow a \mid a.A$

$B \rightarrow b \mid b.B$

$C \rightarrow c \mid c.C$

$a^i b^j c^i$ (attribute grammar)

$S(n) \rightarrow A(n_a).B(n_b).C(n_c) \quad [n=n_a=n_b=n_c]$

$A(n) \rightarrow a \quad [n=1]$

$A(n) \rightarrow a.A(m) \quad [n=m+1]$

$B(n) \rightarrow b \quad [n=1]$

$B(n) \rightarrow b.B(m) \quad [n=m+1]$

$C(n) \rightarrow c \quad [n=1]$

$C(n) \rightarrow c.C(m) \quad [n=m+1]$

Outline

Part I. Translating HTNs to Attribute Grammars

- *Hierarchical Task Networks*
- *Attribute Grammars with Set Attributes (Timelines)*
- *Translation Procedure*

Part II. Validation of Hierarchical Plans using Grammars

- *Parsing of Attribute Grammars*



Outline

Part I. Translating HTNs to Attribute Grammars

- *Hierarchical Task Networks*
- *Attribute Grammars with Set Attributes (Timelines)*
- *Translation Procedure*

Part II. Validation of Hierarchical Plans using Grammars

- *Parsing of Attribute Grammars*



Motivation

Can we represent existing planning domain modeling formalisms as an attribute grammar?

Why?

- provide a unifying framework for modeling planning domains and problems that can be used for **domain model verification, plan and goal recognition, plan validation, domain model acquisition**, as well as for **efficient planning**

Why attribute grammars?

- to exploit existing techniques from formal languages

Automated planning

find a sequence of actions (a plan) to achieve some goal (solve a task)

uses action model with preconditions and effects

Load-r(container, robot, location)

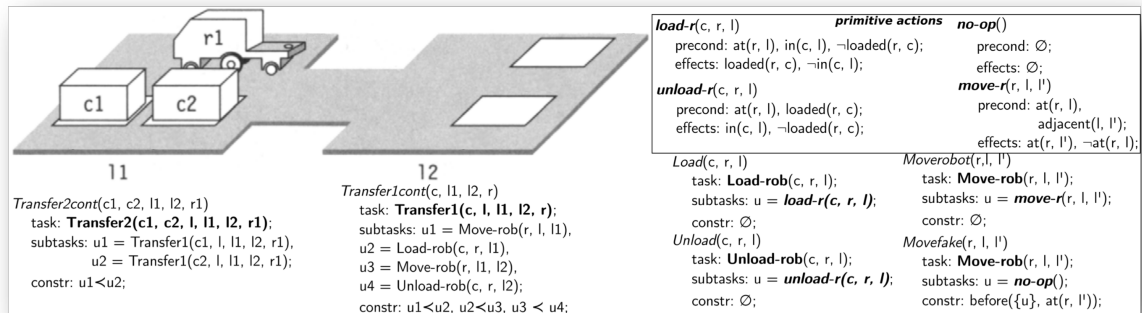
Pre: in(container, location), at(robot, location),
not loaded(robot, container)

Eff: not in(container, location), loaded(robot, container)

HTN

Hierarchical Task Networks seem like a natural candidate to translate to attribute grammars.

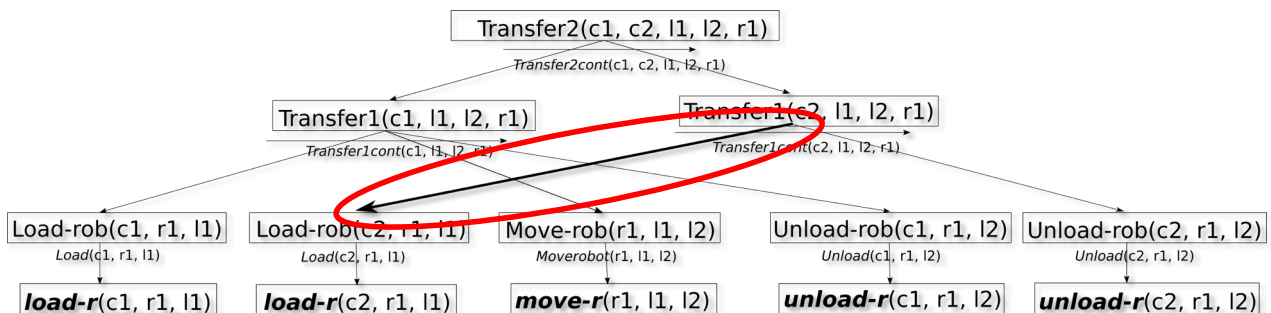
- a hierarchical structure
- some extra ordering constraints



Grammars have been used to represent HTN
Can they cover HTN fully?

HTN – activity interleaving

No existing grammar formalism used to encode HTNs supports task interleaving!



Set attributes

Assume that we want to assign indexes to symbols in $a^i b^j c^k$ such that any permutation of indexes is allowed such as $a_1 b_2 c_3$, $a_1 b_3 c_2$, $a_2 b_1 c_3$ etc.

$$\begin{array}{ll}
 S(n, I) \rightarrow A(n_a, I_a).B(n_b, I_b).C(n_c, I_c) & [n=n_a=n_b=n_c, I = I_a \cup I_b \cup I_c, \\
 & \text{dom}(I, 1, 3n), \text{allDiff}(I)] \\
 A(n, I) \rightarrow a(i) & [n=1, I=\{i\}] \\
 A(n, I) \rightarrow a(i).A(m, I') & [n=m+1, I=\{i\} \cup I'] \\
 B(n, I) \rightarrow b(i) & [n=1, I=\{i\}] \\
 B(n, I) \rightarrow b(i).B(m, I') & [n=m+1, I=\{i\} \cup I'] \\
 C(n, I) \rightarrow c(i) & [n=1, I=\{i\}] \\
 C(n, I) \rightarrow c(i).C(m, I') & [n=m+1, I=\{i\} \cup I']
 \end{array}$$

Timeline constraint

Actions generated during decomposition introduce “before” (precondition) and “after” (effect) events.

The **timeline constraint** ensures the correct ordering of events.

position action	0	1 load-r(c1, r1, l1)	2 load-r(c2, r1, l1)	3 move-r(r1, l1, l2)	4 unload-r(c1, r1, l2)	5 unload-r(c2, r1, l2)
loaded(r1, c1)	a ⁻	b ⁻ a ⁺			b ⁺ a ⁻	
loaded(r1, c2)	a ⁻		b ⁻ a ⁺			b ⁺ a ⁻
at(r1, l1)	a ⁺	b ⁺	b ⁺	b ⁺ a ⁻		
at(r1, l2)	a ⁻			a ⁺	b ⁺	b ⁺
in(c1, l1)	a ⁺	b ⁺ a ⁻				
in(c1, l2)	a ⁻				a ⁺	
in(c2, l1)	a ⁺		b ⁺ a ⁻			
in(c2, l2)	a ⁻					a ⁺

HTN as an attribute grammar

principles

Rewriting rules describe task decompositions, the timeline constraint orders the actions.

Initialisation

$$S(S_0) \rightarrow TN_0(I, TL') \quad [n = |I|, \text{dom}(I, 1, n), \text{allDiff}(I), \\ TL = TL' \cup \text{InitEvents}(S_0), \text{Timeline}(TL)]$$

Task network

$$TN_0(I, TL) \rightarrow T_1(I_1, TL_1) \dots T_m(I_m, TL_m) \quad [C]$$

Task decomposition to task networks

$$T_k(I, TL) \rightarrow TN_{k1}(I, TL) \mid \dots \mid TN_{kn}(I, TL) \quad []$$

Primitive task (action)

$$T_k(I, TL) \rightarrow a_k(i) \quad [I = \{i\}, TL = \text{events}(a_k, i)]$$

HTN as an attribute grammar

example

$$\text{Transfer1}_{c,l,l1,l2,r}(I, TL) \rightarrow \text{Transfer1cont}_{c,l,l1,l2,r}(I, TL)$$

$$\text{Transfer1cont}_{c,l,l1,l2,r}(I, TL) \rightarrow \text{Move-rob}_{r,l,l1}(I_1, TL_1).$$

$$\text{Load-rob}_{c,r,l1}(I_2, TL_2).$$

$$\text{Move-rob}_{r,l1,l2}(I_3, TL_3).$$

$$\text{Unload-rob}_{c,r,l2}(I_4, TL_4) \quad [C]$$

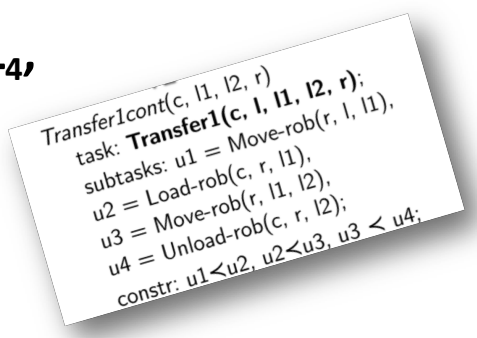
where $C = \{TL = TL_1 \cup TL_2 \cup TL_3 \cup TL_4,$

$$I = I_1 \cup I_2 \cup I_3 \cup I_4,$$

$$\max(I_1) < \min(I_2),$$

$$\max(I_2) < \min(I_3),$$

$$\max(I_3) < \min(I_4)\}$$



```
Transfer1cont(c, l1, l2, r)
task: Transfer1(c, l, l1, l2, r);
subtasks: u1 = Move-rob(r, l, l1),
u2 = Load-rob(c, r, l1),
u3 = Move-rob(r, l1, l2),
u4 = Unload-rob(c, r, l2);
constr: u1 < u2, u2 < u3, u3 < u4;
```

Outline

Part I. Translating HTNs to Attribute Grammars

- *Hierarchical Task Networks*
- *Attribute Grammars with Set Attributes (Timelines)*
- *Translation Procedure*

Part II. Validation of Hierarchical Plans using Grammars

- *Parsing of Attribute Grammars*



Motivation

Validate compliance of a given plan with respect to an HTN model.

Why is it **important**?

- competitions of HTN planners
- soundness of planners and HTN models
- first step towards plan/goal recognition

What is **novel** there?

- the first approach that covers full HTN

Task decomposition as a grammar rule

Transfer1(c, l1, l2, r) \rightarrow Load-rob(c, r, l1).
Move-rob(r, l1, l2).
Unload-rob(c, r, l2) [C]

C = {Load-rob < Move-rob,
Move-rob < Unload-rob,
before({Load-rob}, at(r, l1)),
before({Load-rob}, at(c, l1)),
between({Load-rob},{Move-rob}, at(r, l1)),
between({Move-rob},{Unload-rob}, at(r, l2)),
between({Load-rob},{Unload-rob}, in(c, r))}

HTN plan validation via parsing

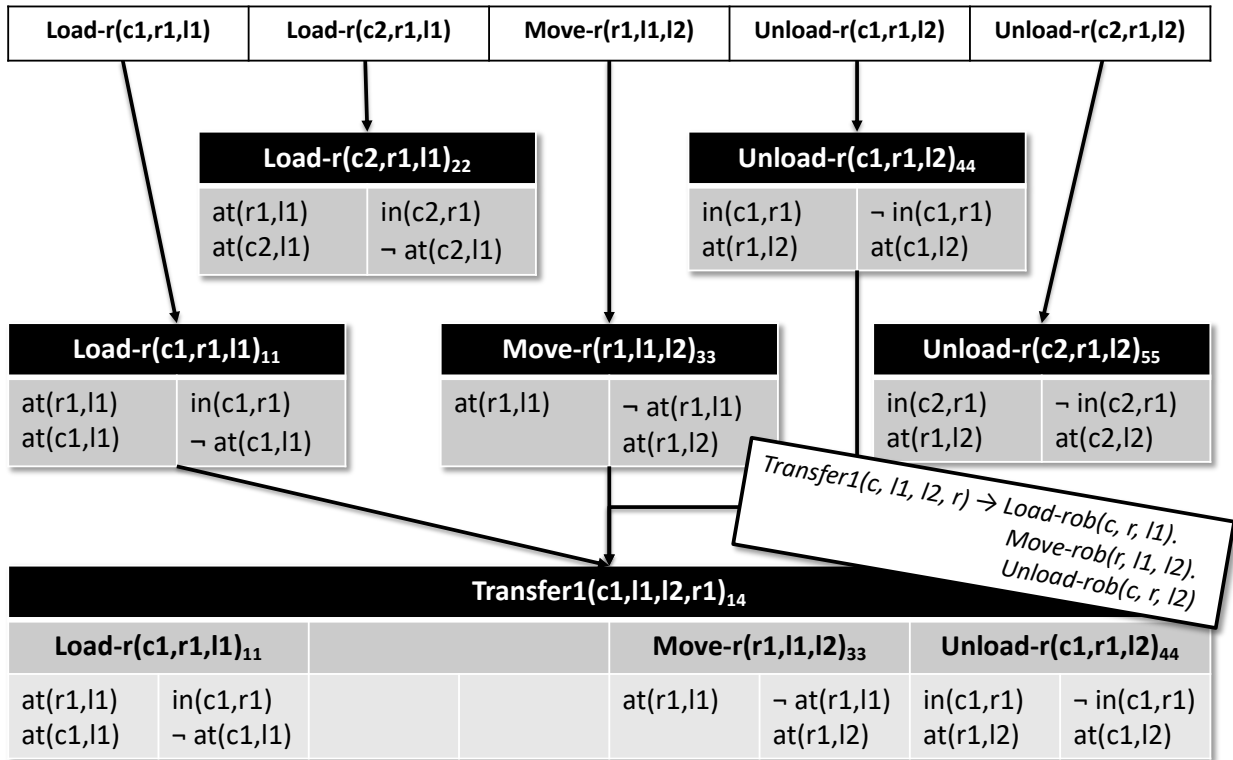
Parsing is a bottom-up approach that groups actions (terminals) to tasks (non-terminals).

For each derived task, we keep a **timeline** – a sequence of slots containing:

- actions (some slots may be empty)
- known effects and partially specified states

Rule is fired when **matching tasks** for the right-hand side are found and their **timelines** can be properly **merged**.

Example of parsing step



Example of parsing step

Timeline merging

Transfer1(c1,l1,l2,r1)₁₄							
Load-r(c1,r1,l1)₁₁				Move-r(r1,l1,l2)₃₃		Unload-r(c1,r1,l2)₄₄	
at(r1,l1)	in(c1,r1)			at(r1,l1)	¬ at(r1,l1)	in(c1,r1)	¬ in(c1,r1)
at(c1,l1)	¬ at(c1,l1)			at(r1,l2)	at(r1,l2)	at(c1,l2)	at(c1,l2)

$before(\{Load-rob\}, at(r1, l1)),$
 $before(\{Load-rob\}, at(c1, l1)),$
 $between(\{Load-rob\}, \{Move-rob\}, at(r1, l1)),$
 $between(\{Move-rob\}, \{Unload-rob\}, at(r1, l2)),$
 $between(\{Load-rob\}, \{Unload-rob\}, in(c1, r))$

Add method constraints

	at(r1,l1)		in(c1,r1)
	in(c1,r1)		in(c1,r1)

Propagate states

	¬ at(c1,l1)		¬ at(r1,l1)
--	-------------	--	-------------

$Pre_{i+1}^+ = (Pre_i^+ \setminus Post_i^-) \cup Post_i^+$
 $Pre_{i+1}^- = (Pre_i^- \setminus Post_i^+) \cup Post_i^-$

The algorithm

```

Data: a set of subplans : subplans
Result: a set of slots newtimeline, the aggregation of the
slots of every subplan
1 Function MERGEPLANS(subplans)
2   lb = min_{(T_i, b_i, e_i, timeline_i) ∈ subplans} b_i;
3   ub = max_{(T_i, b_i, e_i, timeline_i) ∈ subplans} e_i;
4   newtimeline ← {(0, 0, empty, 0, 0)}; i ∈ lb..ub;
5   for (T, b, e, timeline) ∈ subplans do
6     for s_k ∈ timeline, s_k ∈ newtimeline do
7       s_k ← MERGESLOTS(s_k, s_k)
8     end
9   end
10  return newtimeline
11 end

Data: two slots s_1 = (Pre_1^+, Pre_1^-, a_1, Post_1^+, Post_1^-), s_2 =
(Pre_2^+, Pre_2^-, a_2, Post_2^+, Post_2^-)
Result: merged slots
1 Function MERGESLOTS(s_1, s_2)
2   if a_1 = empty or a_2 = empty then
3     Pre^+ = Pre_1^+ ∪ Pre_2^+;
4     Pre^- = Pre_1^- ∪ Pre_2^-;
5     Post^+ = Post_1^+ ∪ Post_2^+;
6     Post^- = Post_1^- ∪ Post_2^-;
7     a = a_1 (if a_2 = empty) or a_2 (if a_1 = empty);
8     return (Pre^+, Pre^-, a, Post^+, Post^-);
9   end
10  break
11 end

Data: a set of slot : slots, a set of before constraints
Result: an updated set of slots
1 Function APPLYPRE(slots, pre)
2   for before(U, l) ∈ pre do
3     id = min{b_i | T_i ∈ U};
4     Pre_id^+ ← Pre_id^+ ∪ l^+;
5     Pre_id^- ← Pre_id^- ∪ l^-
6   end
7 end

Data: a plan P = (a_1, ..., a_n), initial state IniState, a goal
task Goal, an attribute grammar
G = (Σ, N, P, S, A, C)
Result: a Boolean equal to true if the plan can be derived
from the hierarchical structure, false otherwise
Function VERIFYPLAN
/* Initialization of the set of
subplans
subplans ←
{(TP_i, i, i, ((Pre_i^+, Pre_i^-, a_i, Post_i^+, Post_i^-); i)) |
a_i ∈ P, (TP_i → a_i, {pre, post}) ∈ P,
Pre_i^+ = {p | before({a_i}, p) ∈ pre},
Pre_i^- = {p | before({a_i}, -p) ∈ pre},
Post_i^+ = {p | after(a_i, p) ∈ post},
Post_i^- = {p | after(a_i, -p) ∈ post}};
Pre_1^+ ← Pre_1^+ ∪ IniState^+;
Pre_1^- ← Pre_1^- ∪ IniState^-;
while ¬PLANISVALID(subplans, P, Goal) do
for each rule R ∈ P of the form
T_0 → T_1, ..., T_k [-<, pre, btw] such that
subtasks = {(T_i, b_i, e_i, tl_i) | i ∈ 1..k} ⊆
subplans do
verify < from rule R else break;
timeline ← MERGEPLANS(subtasks);
APPLYPRE(timeline, pre);
APPLYBETWEEN(timeline, btw);
PROPAGATE(timeline);
if ∃(Pre^+, Pre^-, a, Post^+, Post^-) ∈
timeline, Pre^+ ∩ Pre^- ≠
∅ ∨ Post^+ ∩ Post^- ≠ ∅ then
break
end
b = min_{(T_i, b_i, e_i, tl_i) ∈ subtasks} b_i;
e = max_{(T_i, b_i, e_i, tl_i) ∈ subtasks} e_i;
subplans ←
subplans ∪ {(T_0, b, e, timeline)};
end
if size of subplans has not increased since the l
iteration then
return false
end
end
return true
end

Data: a set of slots slots
Result: an updated set of slots
1 Function PROPAGATE(slots)
2   lb = min_{(Pre_j^+, Pre_j^-, a_j, Post_j^+, Post_j^-) ∈ slots} j;
3   ub = max_{(Pre_j^+, Pre_j^-, a_j, Post_j^+, Post_j^-) ∈ slots} j - 1;
4   /* Propagation to the right
for i = lb to ub do
5     Pre_{i+1}^+ ← Pre_{i+1}^+ ∪ Post_i^+;
6     Pre_{i+1}^- ← Pre_{i+1}^- ∪ Post_i^-;
7     if a_i ≠ empty then
8       Pre_{i+1}^+ ← Pre_{i+1}^+ ∪ (Pre_i^+ \ Post_i^-);
9       Pre_{i+1}^- ← Pre_{i+1}^- ∪ (Pre_i^- \ Post_i^+);
10    end
11  end
12  /* Propagation to the left
for i = ub downto lb do
13    if a_i ≠ empty then
14      Pre_i^+ ← Pre_i^+ ∪ (Pre_{i+1}^+ \ Post_i^-);
15      Pre_i^- ← Pre_i^- ∪ (Pre_{i+1}^- \ Post_i^+);
16    end
17  end
18 end

Data: a set of slot : slots, a set of between constraints
Result: an updated set of slots
1 Function APPLYBETWEEN(slots, between)
2   for between(U, V, l) ∈ between do
3     s = max{e_i | T_i ∈ U} + 1;
4     e = min{b_i | T_i ∈ V};
5     for id = s to e do
6       Pre_id^+ ← Pre_id^+ ∪ l^+;
7       Pre_id^- ← Pre_id^- ∪ l^-
8     end
9   end
10 end

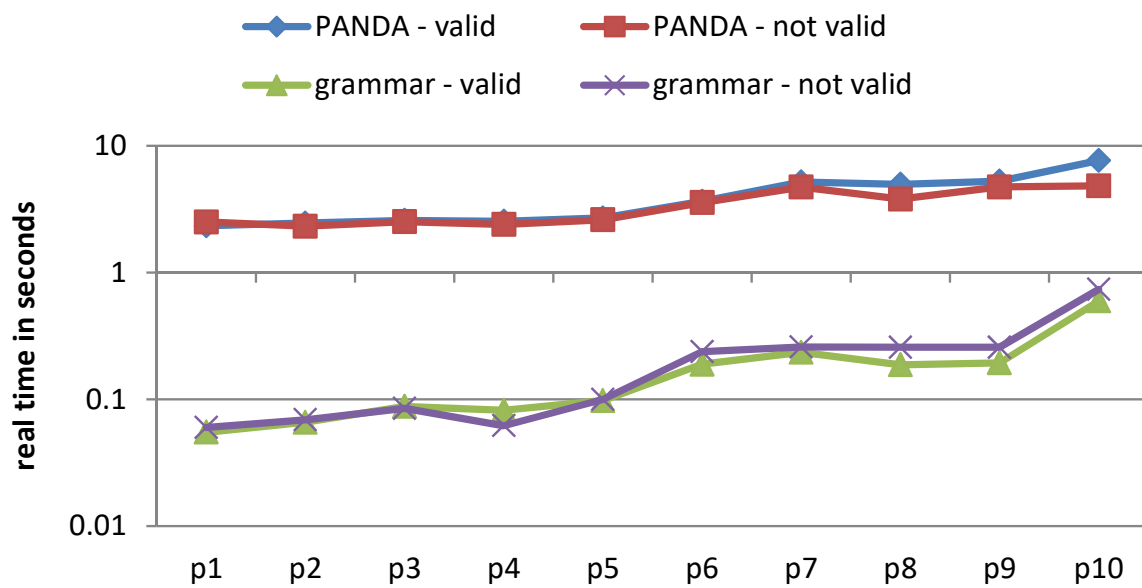
```

Experiments

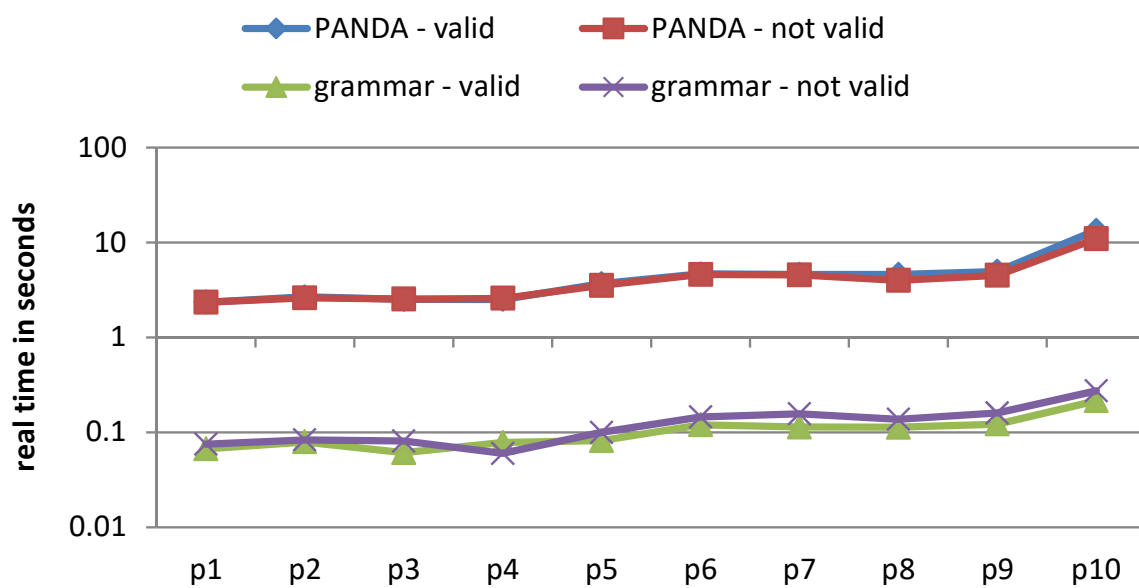
Comparison with PANDA, currently the only HTN plan validator (based on SAT)

- PANDA compiles away some before constraints but cannot handle the other method causal constraints
- Two planning domains
 - Satellite and Transport
- Measuring runtime
 - for correct plans and for wrong plans

Results of experiments - Satellite



Results of experiments - Transport



Summary

STRIPS, HTN with task insertions, and procedural domain control knowledge can be fully **automatically translated** to attribute grammars

- Roman Barták, Adrien Maillard:
Attribute grammars with set attributes and global constraints as a unifying framework for planning domain models. PPDP 2017: 39-48

HTN plans can be **fully validated** with respect to the HTN model using attribute grammars

- Roman Barták, Adrien Maillard, Rafael Cauê Cardoso:
Validation of Hierarchical Plans via Parsing of Attribute Grammars. ICAPS 2018: 11-19

Possible next steps

- identifying a specific bug in plan
- working with partial input plans (plan/goal recognition)
 - plan prefix
 - missing, extra (non-related), wrong observations
 - predicting the next action
- modifying the model to comply with observations
- verifying internal consistency of HTN models



Roman Barták

Charles University, Faculty of Mathematics and Physics
bartak@ktiml.mff.cuni.cz