Automatic Reductions from PH into STRIPS
or
How to Generate Short Problems with Very Long Solutions

Aldo Porco   Alejandro Machado   Blai Bonet

ICAPS. Rome, Italy. June 2013.

UNIVERSIDAD SIMÓN BOLÍVAR

Recap from ICAPS-2011

Introduced a software tool that maps instances of an NP decision problem expressed in SO$\exists$ into a STRIPS problem such that

1) instance is a positive instance iff the STRIPS problem has a plan

2) translation runs in polynomial time

3) STRIPS problem is decidable in non-deterministic polytime (NP)

4) plan, when exists, encodes the solution to input instance

Software Tool

$$\xrightarrow[\sigma,\,\Phi,\,\mathcal{A}]{\text{DCT}} \boxed{\text{tool}} \xrightarrow[\text{dom \& ins}]{\text{PDDL}}$$

Input:

- signature σ that contains relational symbols
- SO$\exists$ formula Φ that encodes NP problem
- finite structure $\mathcal{A}$ that encodes instance

Output:

- PDDLs for a fragment of STRIPS that is decidable in NP

Guarantees:

- runs in polytime for fixed σ and Φ
- output is no harder that input (complexity wise)

Contributions

- Extend tool to target Polynomial Time Hierarchy (PH) instead of only NP
- Generated problems are general STRIPS problems
- Translator runs in polynomial time
- Experimental evaluation over (somewhat) difficult instances

Use:

- Leverage current (planning) technology to NP problems
- Design new benchmarks for planning and test planners and heuristics

Descriptive Complexity Theory (DCT)

Studies complexity theory from a logical perspective without commitments to any model of computation

Major complexity classes had been characterized using different fragments of logic:

- NL is captured by SO$\exists$-Krom (CNF with ≤ 2 literals per clause)
- P is captured by SO$\exists$-Horn (CNF with ≤ 1 positive literal)
- NP is captured by SO$\exists$
- PH is captured by SO
- PSPACE is captured by SO+TC (SO $+$ transitive-closure syntactic construct)

Polynomial Time Hierarchy (PH)

Infinite hierarchy of classes that contains P, NP, $\mathsf{NP}^{\mathsf{NP}}$, $\mathsf{NP}^{\mathsf{NP}^{\mathsf{NP}}}$, etc.

Defined as PH $= \bigcup_{k \geq 0} \Sigma_k^P$ where (using oracles):

- $\Sigma_0^P = \mathsf{P}$
- $\Sigma_1^P = \mathsf{NP}^{\Sigma_0^P} = \mathsf{NP}^{\mathsf{P}} = \mathsf{NP}$
- ...
- $\Sigma_{k+1}^P = \mathsf{NP}^{\Sigma_k^P}$

PH = Co-PH and hence Co-NP and $\Pi_k^P \in$ PH for every $k \geq 0$

It is believed that PH $\neq$ PSPACE; otherwise PSPACE $= \Sigma_k^P$ from some k

Cannonical problem in Σ_k^P is to decide validity of $\exists \bar{x}_1 \forall \bar{x}_2 \exists \bar{x}_3 \cdots \mathcal{Q} \bar{x}_k . \varphi$

Results 1/3

Random formulas of type $\exists\bar{x}\forall\bar{y}\exists\bar{z}.\varphi(\bar{x},\bar{y},\bar{z})$ in Σ_3^P

∃∀∃ #∃	#∀	#∃	n	$+$	$-$	time	len	PDDL in KB
10	2	30	5	—	5	4,199.2	—	17.5
	2	50	5	—	5	2,313.9	—	18.4
30	2	30	5	—	5	3,210.7	—	18.5
	2	50	5	—	5	3,166.3	—	19.4
50	2	30	5	—	1	3,313.4	—	19.4
	2	50	5	3	2	3,450.9	640	20.4

- Random ∃∀∃ problems with 150 clauses each
- Solved with Rintanen's SAT-based planner M
- LAMA'11 does not perform well on this type of problems

Results 2/3

Random instances of $\overline{\text{3Col}}$ in Π_1^P = Co-NP

V	n	$+$	$-$	time / $+$	time / $-$	plan len	PDDL
4	5	1	4	0.1	0.8	1,731	0.4
5	5	2	3	0.6	67.9	6,695	0.6
6	5	2	3	3.4	464.9	26,163	0.7
7	5	2	2	74.8	1.6	102,935	0.8
8	5	1	2	624.0	5.9	406,851	1.0
9	5	—	1	—	0.3	—	1.1

- 5 random graphs for each number of vertices (V)
- Solved with LAMA'11 and obtained very long plans!
- M does not perform well on this type of problems
- **How come does LAMA'11 find a plan with $>$ 400k actions?**

Results 3/3

Random instances of $\overline{\text{3Col}}$ in $\Pi_1^P = \text{Co-NP}$

V	n	$+$	$-$	time / $+$	time / $-$	plan len	PDDL
4	5	1	4	1,850.1	0.1	1,731	0.4
5	5	—	3	—	11.7	—	0.6
6	5	—	3	—	81.9	—	0.7
7	5	—	2	—	0.2	—	0.8
8	5	—	2	—	1.0	—	1.0
9	5	—	1	—	0.0	—	1.1

- The same random instances for $\overline{\text{3Col}}$
- Solved with blind search
- Significantly worse than LAMA'11. Thus, these problems are non-trivial
- **Conjecture:** LAMA'11 solves these instances because of implicit serialization of subgoals enforced by the multiple queues

Example: SAT and UNSAT

Defined over vocabulary $\sigma = \langle P^2, N^2 \rangle$ where:

- $P(x, y)$ tells that variable x appears positive in clause y
- $N(x, y)$ tells that variable x appears negative in clause y

$$\Psi_{\mathsf{SAT}} = \underbrace{(\exists T)}_{\text{s.o. unary relation used to encode guessed assignment}}(\forall y)(\exists x)[(P(x, y) \wedge T(x)) \vee (N(x, y) \wedge \neg T(x))$$

$$\Psi_{\mathsf{UNSAT}} = \underbrace{(\forall T)}_{\text{s.o. unary relation used to encode all assignments}}(\exists y)(\forall x)[(P(x, y) \Rightarrow \neg T(x)) \wedge (N(x, y) \Rightarrow T(x))$$

Example: SAT

$$\Psi_{\mathsf{SAT}} = (\exists T)(\forall y)(\exists x)[(P(x,y) \wedge T(x)) \vee (N(x,y) \wedge \neg T(x))$$

Instance: $\underbrace{(x_0 \vee \neg x_1 \vee x_2)}_{\text{clause 0}} \wedge \underbrace{(\neg x_0 \vee \neg x_2)}_{\text{clause 1}} \wedge \underbrace{(\neg x_0 \vee x_1)}_{\text{clause 2}}$

Instance is satisfiable with model $\{\neg x_0, \neg x_1, \neg x_2\}$

STRIPS plan:

```
(begin-proof)
(end-guess-T)

(prove-and-2 var0 var1)
(prove-or-2 var0 var1)
(prove-exists var0)
(prove-forall-base var0)

(prove-and-2 var1 var0)
(prove-or-2 var1 var0)
(prove-exists var1)
(prove-forall-induc var0 var1)

(prove-and-2 var2 var0)
(prove-or-2 var2 var0)
(prove-exists var2)
(prove-forall-induc var1 var2)

(prove-so-exist-T var2)
(prove-goal)
```

Example: UNSAT

$$\Psi_{\mathsf{UNSAT}} = (\forall T)(\exists y)(\forall x)[(N(x,y) \Rightarrow T(x)) \wedge (P(x,y) \Rightarrow \neg T(x))]$$

Instance: $\underbrace{(x_0)}_{\text{clause 0}} \wedge \underbrace{(\neg x_0 \vee \neg x_1)}_{\text{clause 1}} \wedge \underbrace{(\neg x_0 \vee x_1)}_{\text{clause 2}}$

Instance is unsatisfiable

STRIPS plan:

```
(begin-proof)
(init-so-forall-T)

(prove-or_1_1 var0 var0)
(prove-or_2_2 var0 var0)
(prove-and var0 var0)
(prove-forall_base var0 var0)
(prove-or_1_1 var0 var1)
(prove-or_2_2 var0 var1)
(prove-and var0 var1)
(prove-forall_induc var0 var0 var1)
(prove-exists var0 var1)

(change_for_coin_T var0)
(zero_plus_one_T var0)

(prove-or_1_2 var2 var0)
(prove-or_2_1 var2 var0)
(prove-and var2 var0)
(prove-forall_base var2 var0)
(prove-or_1_1 var2 var1)
(prove-or_2_2 var2 var1)
(prove-and var2 var1)
(prove-forall_induc var2 var0 var1)
(prove-exists var2 var1)

(change_for_coin_T var0)
(one_plus_one_0_T var0 var1)
(zero_plus_one_T var1)

(prove-or_1_1 var0 var0)
(prove-or_2_2 var0 var0)
(prove-and var0 var0)
(prove-forall_base var0 var0
(prove-or_1_2 var0 var1)
(prove-or_2_1 var0 var1)
(prove-and var0 var1)
(prove-forall_induc var0 var0 var1)
(prove-exists var0 var1)

(change_for_coin_T var0)
(zero_plus_one_T var0)
(prove-or_1_2 var1 var0)
(prove-or_2_1 var1 var0)
(prove-and var1 var0)
(prove-forall_base var1 var0)
(prove-or_1_2 var1 var1)
(prove-or_2_1 var1 var1)
(prove-and var1 var1)
(prove-forall_induc var1 var0 var1)
(prove-exists var1 var1)

(change_for_coin_T var0)
(one_plus_one_0_T var0 var1)
(one_plus_one_final_T var1)

(prove-goal)
```

Idea of Translation

For FO formulas (from ICAPS-11):

- fluents represent validity of subformulas where parameters stand for free variables
- operators establish validity of formulas (fluents) from validity of subformulas (inductively in structure of formulas)

For SO existential quantifiers (similar to ICAPS-11):

- plan chooses one interpretation of quantified symbol
- plan then moves and proves validity with chosen interpretation

For SO universal quantifiers:

- plan iterates over all interpretations of quantified symbol
- for each such interpretation, plan proves validity

Iteration over All Interpretations

Consider a unary relation T and a universe with n objects

There are 2^n different interpretations of T that can be identified with the 2^n differents binary words of length n:

i-th element belongs to T's interpretation iff i-th bit in word is 1

Iterating over interpretations is done by iterating over such words

The word is treated as a counter that starts at '$00 \cdots 0$' and is incremented until '$11 \cdots 1$' by adding 1

How to Capture PSPACE

PSPACE $=$ SO $+$ TC

TC$[\Psi](\bar{x}, \bar{y})$ denotes connectivity on a graph defined by the formula $\Psi(\bar{u}, \bar{v})$

If we use the fluent $\mathfrak{F}[\Psi](\bar{u}, \bar{v})$ to denote the validity of $\Psi(\bar{u}, \bar{v})$, then can design actions to prove the validity of TC$[\Psi](\bar{x}, \bar{y})$

Therefore, implementing TC$[\Psi]$ in STRIPS is straightforward:

it is just finding a path on a graph whose edges are given by fluents $\mathfrak{F}[\Psi](\bar{u}, \bar{v})$

Summary

– Extended the tool presented in ICAPS-11 so that:

- targets the much bigger complexity class PH
- implements a type system that permits more efficient translations

– Performed experiments and got interesting results for LAMA'11

– Tool can be used to design challenging benchmarks for planners

– Tool can be extended to target whole PSPACE without much work

Thanks. Questions?