Challenges in Finding Generalized Plans

Siddharth Srivastava
Joint work with Neil Immerman and Shlomo Zilberstein
University of Massachusetts, Amherst

ICAPS 2009 Workshop on
Generalized Planning: Macros, Loops, Domain Control
September 20^{th}, 2009

Generalized Planning

Plans or planning structures that "work in many situations"

- Triangle Tables [Fikes et al., 1972]
- Case Based Planning [Hammond, 1986]
- Explanation Based Planning [Minton et al., 1989, Shavlik, 1990]
- Contingent Planning
- Learning domain specific planners from examples [Winner and Veloso, 2003]; Planning with loops [Levesque, 2005];

Overview

- Universal Challenges
- Our Framework
- Generalized Planning with Sensing Actions
- Results

Examples of Generalized Plans

Classical Plans

mvToTable(b_3), mvToTable(b_2), mvToTable(b_1)

Examples of Generalized Plans

Classical Plans

mvToTable(b_3), mvToTable(b_2), mvToTable(b_1)

More General:

"Unstack":

while($\exists b$: topmost(b)$\wedge\neg$ onTable(b)) $\{$mvToTable(b)$\}$

Examples of Generalized Plans

Classical Plans

`mvToTable(`b_3`)`, `mvToTable(`b_2`)`, `mvToTable(`b_1`)`

More General:

"Unstack":

`while(`$\exists b$`: topmost(b)`$\wedge\neg$` onTable(b))` $\{$`mvToTable(b)`$\}$

Still More General:

FF, SATPLAN, SGPLAN, ...

Examples of Generalized Plans

Classical Plans

`mvToTable(`b_3`), mvToTable(`b_2`), mvToTable(`b_1`)`

More General:

"Unstack":

`while(`$\exists b$`: topmost(b)`$\wedge\neg$` onTable(b)) {mvToTable(b)}`

Still More General:

FF, SATPLAN, SGPLAN, ...

Common fundamental problem (*Generalized Planning*):
Find a function G (a *generalized plan*):

$$G : \text{Problem instance} \longrightarrow \text{sequence of actions}$$

What makes us prefer one over another?

Challenges for Any Approach to Generalized Planning

1. Applicability Test
2. Cost of Instantiation
3. Domain Coverage
4. Quality of instantiated plans
5. Complexity of creating generalized plans

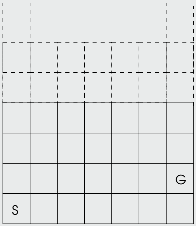
Applicability Test

$$G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots a_n$$

- One approach: simulated execution.
- Cost of instantiation will be wasted if G cannot solve it.

NavigateGrids /*Start at bottom left*/

repeat
  while ¬*rightmost* **do**
    mvR()
  end
  mvU()
  while ¬*leftmost* **do**
    mvL()
  end
  mvU()
until *atgoal*



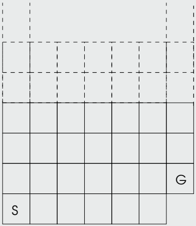
Applicability Test

$$G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots a_n$$

- One approach: simulated execution.
- Cost of instantiation will be wasted if G cannot solve it.

NavigateGrids /*Start at bottom left*/

repeat
  while ¬*rightmost* **do**
    mvR()
  end
  mvU()
  while ¬*leftmost* **do**
    mvL()
  end
  mvU()
until *atgoal*



Applicability Test (ctd.)

- Historically not common: not required for very general (FF) or very simple plans $(a_1, \ldots a_n)$.
- Computed generalized plans typically have a limited applicability.
- More of a problem with compact representations (loops).
  - Simulated execution may not even terminate!!

Ideal applicability test: linear in the size of the problem

Cost of Plan Instantiation

$$G : \text{Problem instance} \xrightarrow{\text{plan instantiation}} a_1, \ldots a_n$$

- Makes generalized plans like "unstack" (O(n)) more desirable than classical planners (O(exp(n))).
- In hindsight: low COI = one of the main motivations behind this field.

Domain Coverage

The set/fraction of solvable problems solved by a generalized plan.

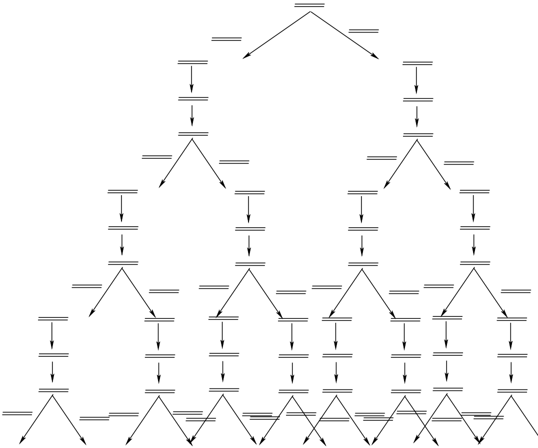
- Historically one of the most measured attributes.
- Trade-offs with cost of instantiation.

Quality of Instantiated Plans

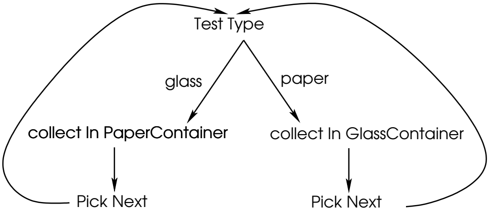
The computational cost (makespan/number of actions/time etc.) of executing the instantiated plan.

- Satisficing, optimal generalized plans.
- Trade-offs with domain coverage and cost of instantiation.

Complexity of Creating Generalized Plans



Complexity of Creating Generalized Plans



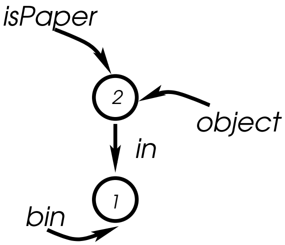
- Serious problems with applicability test, instantiation:
  - Loop termination, progress

Our Objective

- Compute algorithm-like “generalized” plans.
    - Low cost of instantiation
    - Efficient applicability tests
    - Efficient generation of generalized plans
- Need to determine progress and termination.

Concrete States as Logical Structures

$$\mathcal{V} = \{object^1, bin^1, isGlass^1, isPaper^1, in^2, empty^1, collected^1, forGlass^1, forPaper^1\}$$



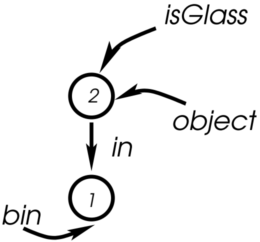
$[\![object(2)]\!] = 1$

$[\![isPaper(2)]\!] = 1$

$[\![bin(1)]\!] = 1$

$[\![in(2,1)]\!] = 1$

S_1



$[\![object(2)]\!] = 1$

$[\![isGlass(2)]\!] = 1$

$[\![bin(1)]\!] = 1$

$[\![in(2,1)]\!] = 1$

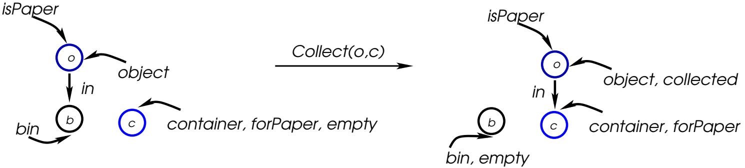
S_2

Example: The Collect Action

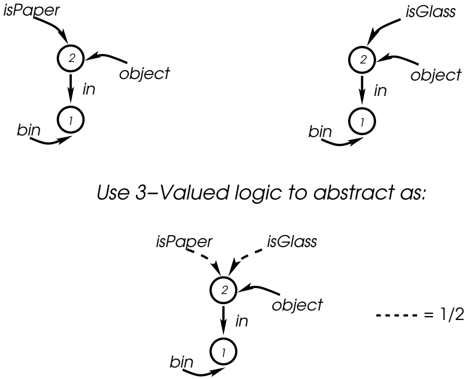
Collect(o,c)

$\mathrm{object}(o) \wedge \mathrm{container}(c) \wedge (\mathrm{isGlass}(o) \leftrightarrow \mathrm{forGlass}(c)) \wedge$
$\exists b(\mathrm{bin}(b) \wedge \mathrm{in}(o,b) \wedge \mathrm{robotAt}(b))$

$$\begin{aligned}
in'(u,v) &:= (in(u,v) \wedge u \neq o) \vee \\
&\quad\; (\neg in(u,v) \wedge u = o \wedge v = c) \\
empty'(u) &:= (empty(u) \wedge u \neq c) \vee in(o,u) \\
collected'(u) &:= collected(u) \vee o = u
\end{aligned}$$

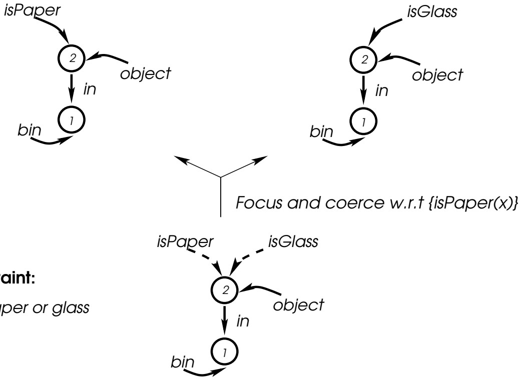


Abstraction Using 3-Valued Logic



TVLA: [Sagiv et al., 2002]

Abstraction Using 3-Valued Logic

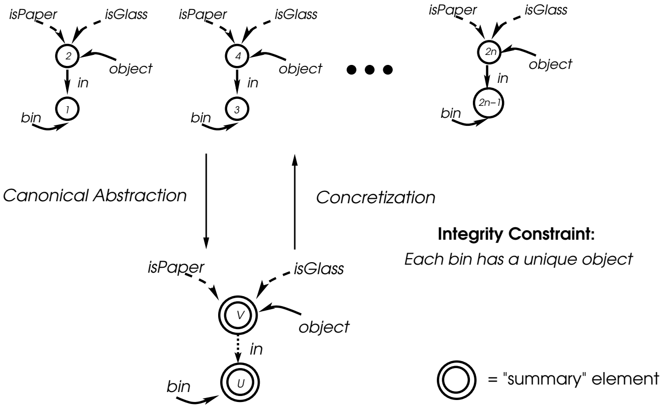


Integrity Constraint:

Objects are either paper or glass

Implementation of "sensing" actions

Abstraction Using 3-Valued Logic



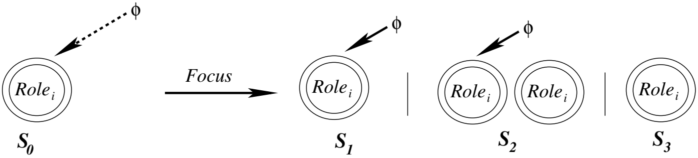
Abstraction Using 3-Valued Logic: Summary

TVLA [Sagiv et al., 2002]: Three Valued Logic Analysis

- Abstraction predicates: unary predicates.
- Element's role = set of abstraction predicates satisfied
- Collapse elements of a role into summary elements.
- Use integrity constraints to retreive concrete states.

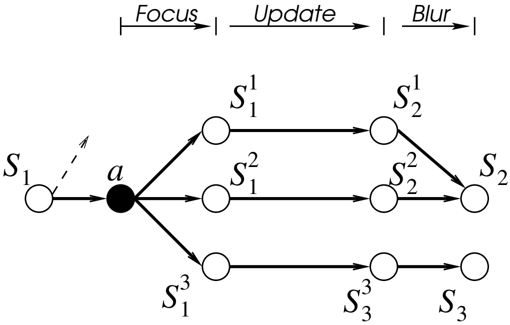
Action Application on Belief States

- Make structures precise by creating possible cases: focus (automatic)
- Apply action



Action Application on Belief States

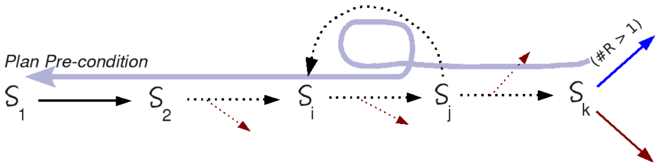
- Make structures precise by creating possible cases: focus (automatic)
- Apply action



Action Branches and Plan Preconditions

Branches solve only *some* members of abstract structures

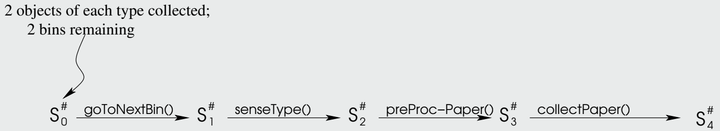
- May be classifiable, e.g $\#_R\{S\} > 1$
  - Extended-LL domains: all branches are classifiable
- Subtract role-count changes to obtain preconditions at start.
- Generalize to simple loops, nested loops due to shortcuts and sensing actions.



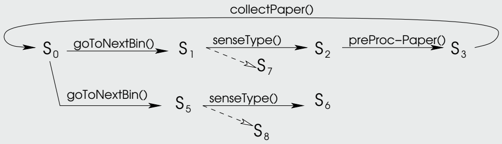
Plan Generalization

Use abstract structures to recognize loop invariants in example concrete plans.

Example Execution



Find Loops



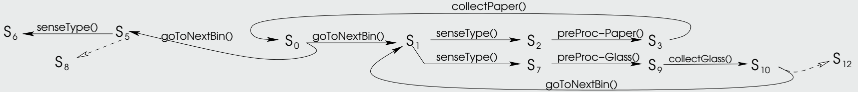
Developed for completely observable settings [Srivastava et al., 2008]

Merging Generalized Plans

Plan for Unhandled Structure

$$S_7^{\#} \xrightarrow{\text{preProc-Glass()}} S_9^{\#} \xrightarrow{\text{collectGlass()}} S_{10}^{\#} \xrightarrow{\text{goToNextBin()}} S_{11}^{\#} - - - - -$$

Generalize and Merge

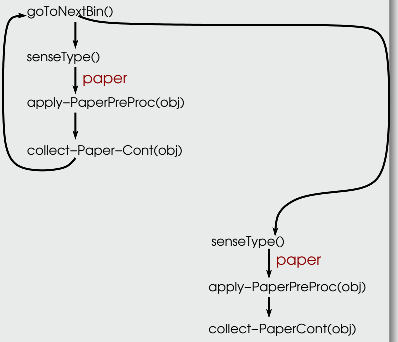


- A single plan may not explore all possibilities.
- Construct problem instances from unsolved belief states.
- Solve them using classical planners.

Example Results

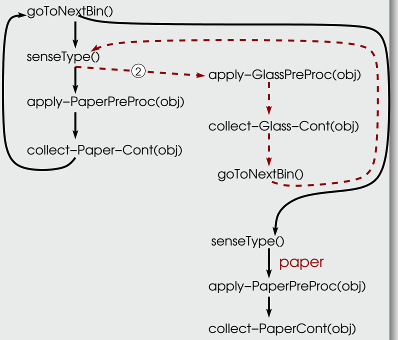
$p_0 = \|\{\text{paper, collected}\}\|; pc_0 = \|\{\text{empty,container,forPaper}\}\|;$
g_0, gc_0 : similar for glass; $b_0 = \|\{\text{bin}\}\|$

Loop 1



- Precons: $pc_0 = l_1; b_0 = l_1$
- Solves 1 out of 2^n

Loops 1 & 2



- Precons: $pc_0 = l_1; gc_0 = l_2; b_0 = l_1 + l_2$
- $2^{n-1} + 1$ out of every 2^n

Merging Generalized Plans: Algorithm

Input: Existing plan Π, eg trace trace_i
Output: Extension of Π
if $\Pi = \emptyset$ **then**
    $\Pi \leftarrow \mathrm{trace}_i$
    return Π
end
$\mathrm{mp}_\Pi, \mathrm{mp}_t \leftarrow$ findMergePoint$(\Pi, \mathrm{trace}_i, \mathrm{bp}_\Pi, \mathrm{bp}_t)$
repeat
    if *mp_Π found* **then**
        $\mathrm{bp}_\Pi, \mathrm{bp}_t \leftarrow$ findBranchPoint$(\Pi, \mathrm{trace}_i, \mathrm{mp}_\Pi, \mathrm{mp}_t)$
    end
    if *bp_Π found* **then**
        $\mathrm{mp}_\Pi, \mathrm{mp}_t \leftarrow$ findMergePoint$(\Pi, \mathrm{trace}_i, \mathrm{bp}_\Pi, \mathrm{bp}_t)$
        addEdges$(\Pi, \mathrm{trace}_i, bp_t, mp_t, mp_\Pi, bp_\Pi)$
    end
until *new bp_Π or mp_Π not found*
return Π

Algorithm 1: ARANDA-Merge

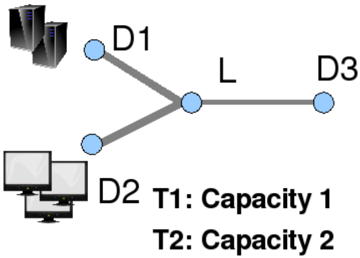
Addressing the Challenges

- Cost of testing applicability: independent of the size of the problem.
- Cost of instantiation: linear, or better with role-lists
- Domain Coverage can increase exponentially with new examples
- Complexity of creating generalized plan: $O(s \cdot n_{eg}^2)$ to find loops, $O(s \cdot n_{eg})$ for preconditions.

Conclusions

- Clear formal framework for algorithmic plans, avoiding intractability of automated program synthesis.
- Approach for learning generalized conditional plans with nested loops by composition of simple linear plans.
- Efficient methods for computation of measures of progress and preconditions.

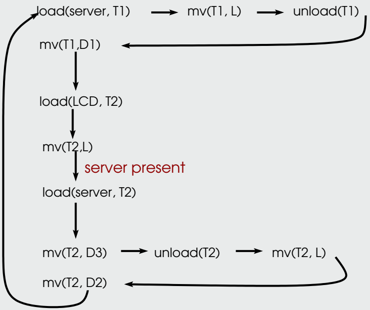
Transport Domain



Transport Domain: Results

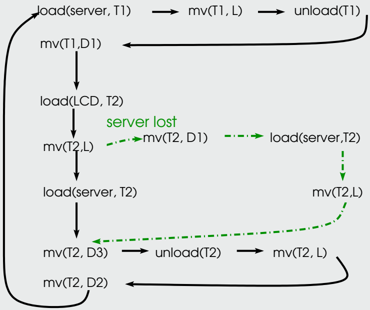
$$m_0 = \|\{\text{monitor, atD2}\}\|; s_0 = \|\{\text{server, atD1}\}\|$$

Loop 1



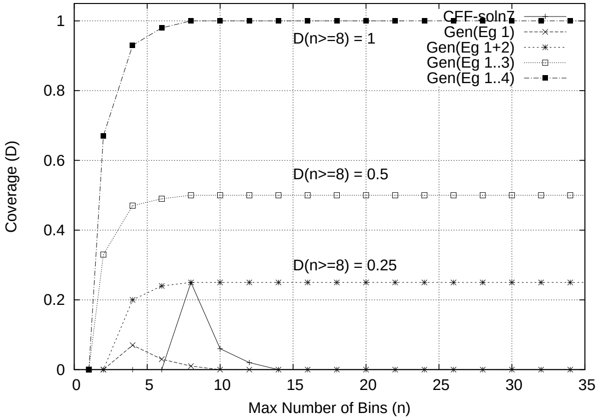
- Precons: $m_o = l_1; s_0 = l_1$

Loops 1 & 2



- Precons: $m_0 = l_1; s_o = l_1 + k_1$

Example Results: Domain Coverage



$$D_\pi(n) = |\mathcal{S}_\pi(n)|/|\mathcal{T}(n)|$$

Related Work

- Plans with Loops
  - [Winner and Veloso, 2007]: no preconditions or sensing actions, but use partial ordering.
  - [Levesque, 2005]: single planning parameter, limited preconditions.
  - [Cimatti et al., 2003]: "hard" loops.
- Planning with unknown quantities:
  - [Milch et al., 2005]: action operators not provided.

References I

- Cimatti, A., Pistore, M., Roveri, M., and Traverso, P. (2003).
  Weak, strong, and strong cyclic planning via symbolic model checking.
  Artif. Intell., 147(1-2):35–84.
- Fikes, R., Hart, P., and Nilsson, N. (1972).
  Learning and Executing Generalized Robot Plans.
  Technical report, AI Center, SRI International.
- Hammond, K. (1986).
  CHEF: A Model of Case-Based Planning.
  In *Proceedings of AAAI-86*, pages 267–271.
- Levesque, H. J. (2005).
  Planning with loops.
  In *Proc. of IJCAI*, pages 509–515.

References II

Milch, B., Marthi, B., Russell, S. J., Sontag, D., Ong, D. L., and Kolobov, A. (2005).
Blog: Probabilistic models with unknown objects.
In *Proc. of IJCAI*, pages 1352–1359.

Minton, S., Carbonell, J. G., Knoblock, C. A., Kuokka, D. R., and Etzioni, O. (1989).
Explanation-based Learning: A Problem Solving Perspective.
Artif. Intell., 40(1-3).

Sagiv, M., Reps, T., and Wilhelm, R. (2002).
Parametric shape analysis via 3-valued logic.
ACM Transactions on Programming Languages and Systems, 24(3):217–298.

References III

Shavlik, J. W. (1990).
Acquiring recursive and iterative concepts with explanation-based learning.
Machine Learning, 5:39–40.

Srivastava, S., Immerman, N., and Zilberstein, S. (2008).
Learning generalized plans using abstract counting.
In *Proc. of AAAI*, pages 991–997.

Winner, E. and Veloso, M. (2003).
Distill: Learning domain-specific planners by example.
In *Proc. of ICML*, pages 800–807.

References IV

Winner, E. and Veloso, M. (2007).
LoopDISTILL: Learning domain-specific planners from example plans.
In *Workshop on AI Planning and Learning, ICAPS*.