Finding Plans with Branches, Loops, and Preconditions

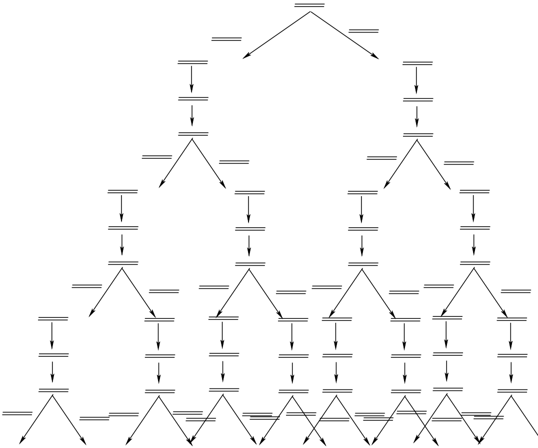
Siddharth Srivastava
Joint work with Neil Immerman and Shlomo Zilberstein
University of Massachusetts, Amherst

ICAPS 2009 Workshop on
Verification and Validation of Planning and Scheduling Systems
September 20^{th}, 2009

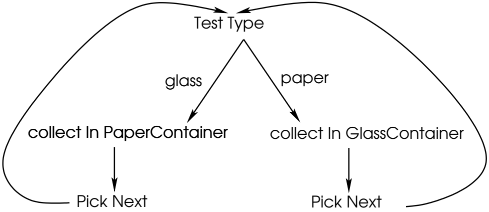
Overview

- Introduction
- Our Framework
- Planning Algorithms
- Results

Conditional Planning



Conditional Planning



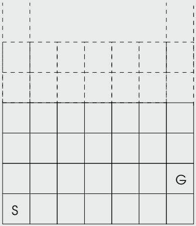
- Serious problems with applicability test, instantiation:
  - Loop termination, progress

Plan Preconditions

- One approach: simulated execution.
- Will be wasted if G cannot solve a problem.

NavigateGrids /*Start at bottom left*/

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

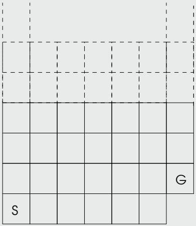


Plan Preconditions

- One approach: simulated execution.
- Will be wasted if *G* cannot solve a problem.

NavigateGrids /*Start at bottom left*/

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



Plan Preconditions (ctd.)

- Historically not common: not required for simple plans $(a_1, \ldots a_n)$.
- Computed plans with loops etc. will typically have a limited applicability.
  - Simulated execution may not even terminate!!

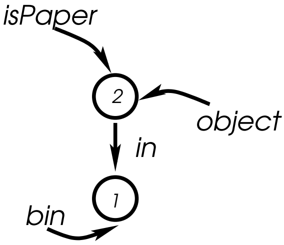
Ideal applicability test: linear in the size of the problem

Our Objective

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

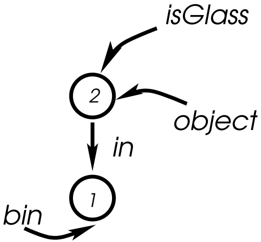
Concrete States as Logical Structures

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



((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

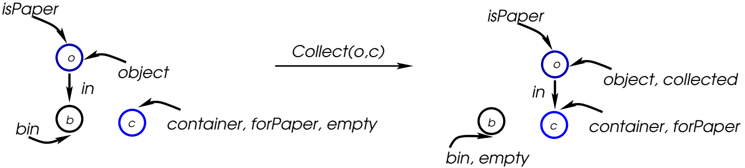
S_2

Example: The Collect Action

Collect(o,c)

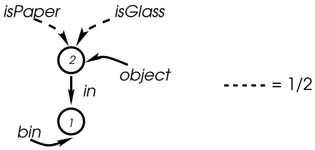
$\text{object}(o) \wedge \text{container}(c) \wedge (\text{isGlass}(o) \leftrightarrow \text{forGlass}(c)) \wedge \exists b(\text{bin}(b) \wedge \text{in}(o,b) \wedge \text{robotAt}(b))$

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



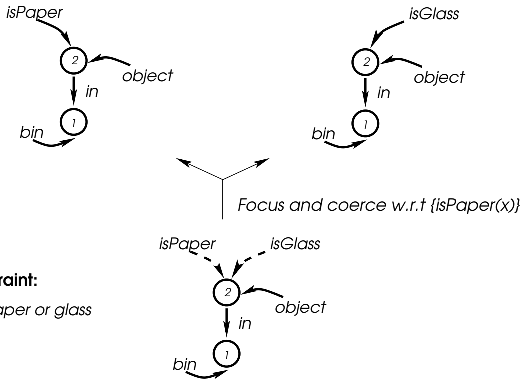
Abstraction Using 3-Valued Logic

Use 3–Valued logic to abstract as:



TVLA: [Sagiv et al., 2002]

Abstraction Using 3-Valued Logic

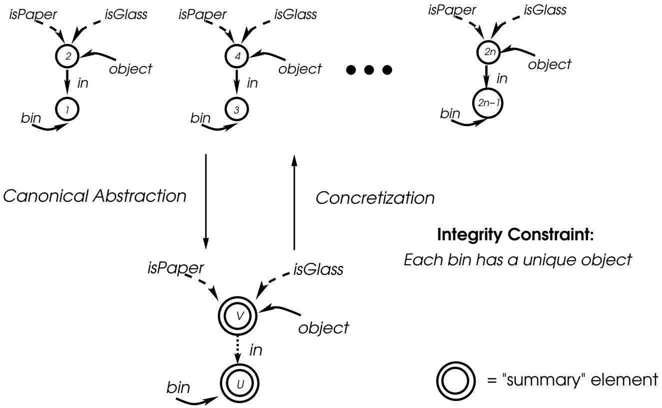


Integrity Constraint:

Objects are either paper or glass

Implementation of "sensing" actions

Abstraction Using 3-Valued Logic



Integrity Constraint:

Each bin has a unique object

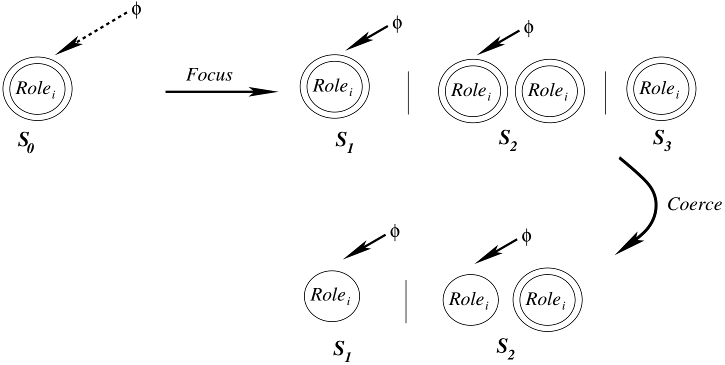
Abstraction Using 3-Valued Logic: Summary

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.

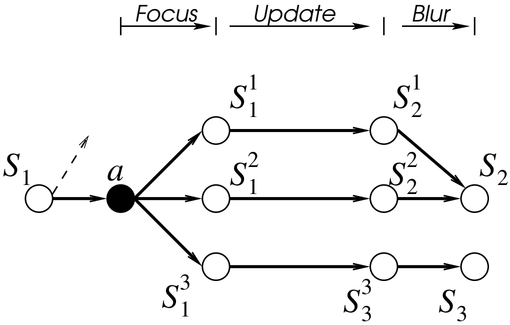
Action Application on Belief States

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

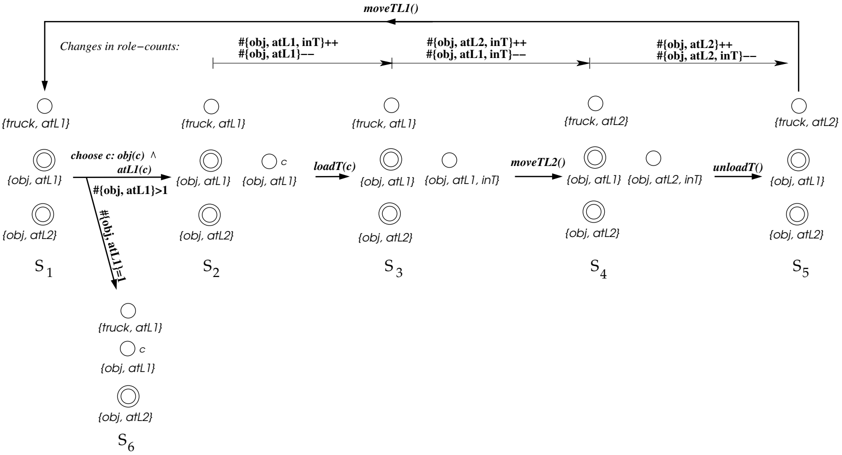


Action Application on Belief States

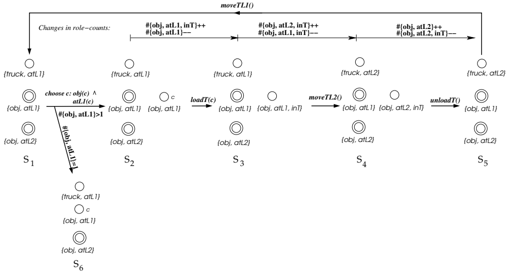
- Make structures precise by creating possible cases: focus (automatic)
- Apply action



Role-counts, Branches and Plan Preconditions



Role-counts, Branches and Plan Preconditions

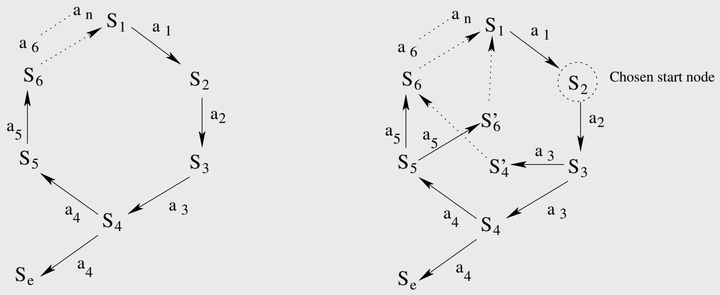


- Goal is provably reachable from the infinitely many structures represented by S_1.
- $\forall s \in S_1$, can compute number of steps required to reach the goal.

Generalized to *extended-LL* domains.

Extension to Complex Loops with Shortcuts

Shortcuts due to sensing actions



(a) A simple loop (b) A simple loop with (non−composable) shortcuts

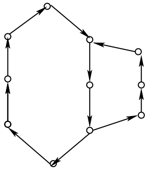
Extension to Nested Loops?

```
for i = 1 to n {
    for j = 1 to k {

    }

}
```

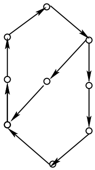
```
for i = 1 to n {

    if (...) {

    }
    if (...) {

    }
}
```

Extension to Nested Loops?

```
for i = 1 to n {
    for j = 1 to k {
        ...
    }
    ...
}
```

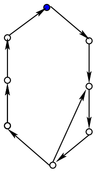


```
for i = 1 to n {
    ...
    if (...) {
        ...
    }
    if (...) {
        ...
    }
}
```

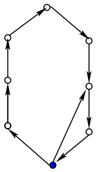


Extension to Nested Loops?

```
for i = 1 to n {
    a1
    for j = 1 to k {
        a2
    }
    a3
}
```



```
i = 1; j = 1
if (i < n+1) { a1 }
for i = 1 to n {
    if (j > k) {
        a3
        i++
        a1
        j = 1
    }
    if (j < k+1) {
        a2
        j++
    }
}
```

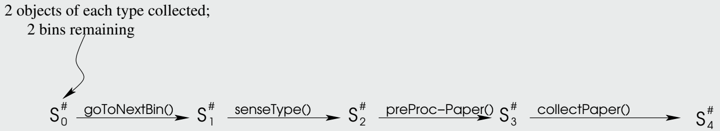


- Execution sequences match.
- Translation of the loop entry point $\implies$ complex loop with a shortcut!
- Methods applicable to many so-called “nested” loops.

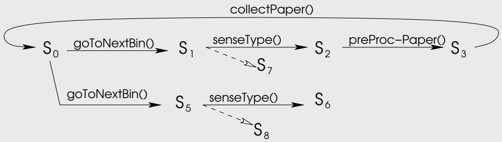
Plan Generalization

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

Example Execution



Find Loops



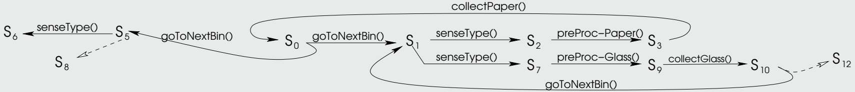
Developed for completely observable settings [Srivastava et al., 2008]

Merging Generalized Plans

Plan for Unhandled Structure

$$S^{\#}_{7} \xrightarrow{\text{preProc-Glass()}} S^{\#}_{9} \xrightarrow{\text{collectGlass()}} S^{\#}_{10} \xrightarrow{\text{goToNextBin()}} S^{\#}_{11} \text{- - - - -}$$

Generalize and Merge

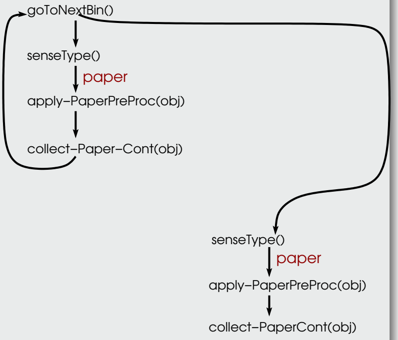


- A single plan may not explore all possibilities.
- Construct problem instances from unsolved belief states.
- Solve them using classical planners.

Example Results

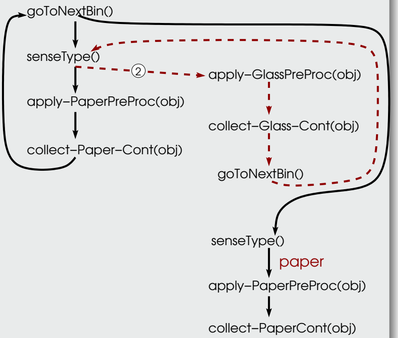
$p_0 = \|\{\text{paper, collected}\}\|; pc_0 = \|\{\text{empty,container,forPaper}\}\|;$
g_0, gc_0 : similar for glass; $b_0 = \|\{\text{bin}\}\|$

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

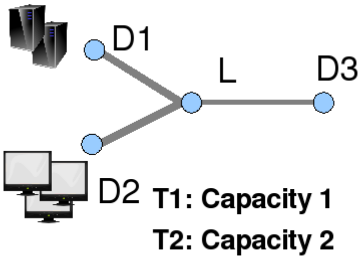
if $\Pi = \emptyset$ **then**
    $\Pi \leftarrow \text{trace}_i$
    return Π
end
$\text{mp}_\Pi, \text{mp}_t \leftarrow$ findMergePoint$(\Pi, \text{trace}_i, \text{bp}_\Pi, \text{bp}_t)$
repeat
    if *mp_Π found* **then**
        $\text{bp}_\Pi, \text{bp}_t \leftarrow$ findBranchPoint$(\Pi, \text{trace}_i, \text{mp}_\Pi, \text{mp}_t)$
    end
    if *bp_Π found* **then**
        $\text{mp}_\Pi, \text{mp}_t \leftarrow$ findMergePoint$(\Pi, \text{trace}_i, \text{bp}_\Pi, \text{bp}_t)$
        addEdges$(\Pi, \text{trace}_i, bp_t, mp_t, mp_\Pi, bp_\Pi)$
    end
until *new bp_Π or mp_Π not found*
return Π

Algorithm 1: ARANDA-Merge

Conclusions

- Approach addressing plan/algorithm synthesis and verification.
  - Advantage of automated synthesis: can choose to keep control structure verifiable.
- Close to program synthesis, but free of associated intractability.
- Efficient precondition tests and measures of progress.

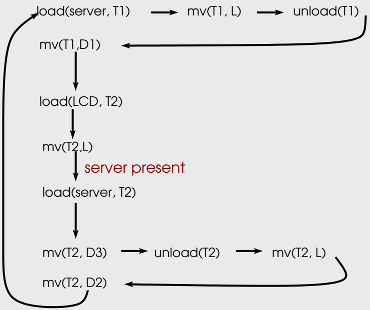
Transport Domain



Transport Domain: Results

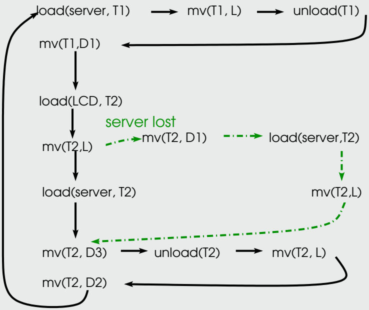
$$m_0 = \|\{\text{monitor, atD2}\}\|; s_0 = \|\{\text{server, atD1}\}\|$$

Loop 1



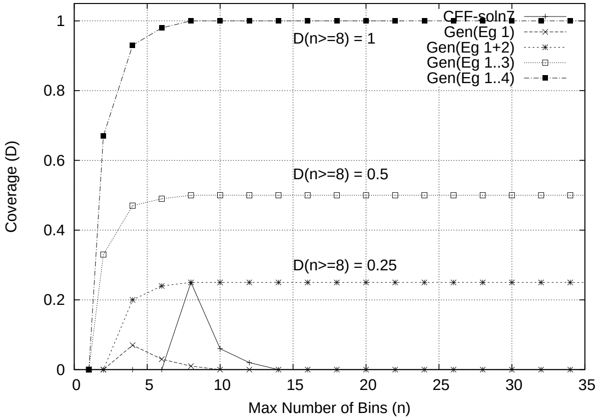
- Precons: $m_o = l_1; s_0 = l_1$

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.

- Levesque, H. J. (2005).
  Planning with loops.
  In *Proc. of IJCAI*, pages 509–515.

- 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.

References II

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