Learning Generalized Plans Using Abstract Counting

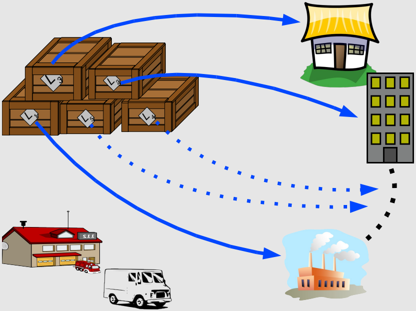
Siddharth Srivastava, Neil Immerman, Shlomo Zilberstein

Twenty Third Conference on Artificial Intelligence
17^{th} July, 2008

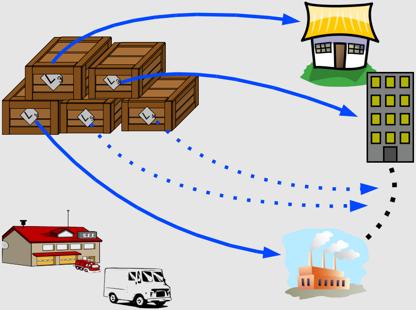
Overview

- Introduction
- Our Approach
  - Abstraction Mechanism
  - Algorithm for Learning Generalized Plans
- Results
- Conclusions

Plans vs Algorithms



Plans vs Algorithms



Move Truck to Dock

While #(undelivered crate)>0

Load a crate

Find crate's destination

Move truck to destination

Unload crate

Move Truck to Dock

Move Truck to Garage

Finding Algorithm-like Plans

Variants of this problem have been of continued interest.

Recurring Hurdles

- Problem definition: unknown numbers
- Plans with loops: finding loops
- Plans with loops: reasoning about loops (*Plan correctness*)

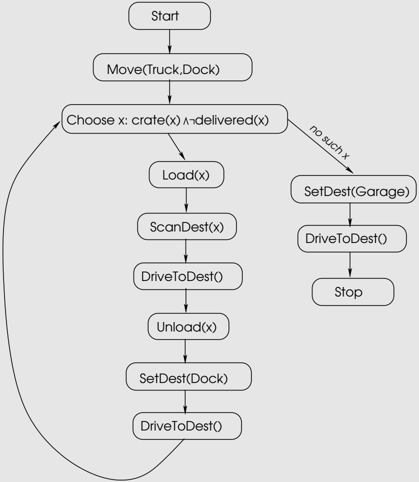
Myth Systematic approach $\Longrightarrow$ undecidable
(cf. automated programming)

We identify a tractable piece of this problem.

Generalized Plans

A formalization of algorithm-like plans.

- Connected, directed graph.
- Nodes → actions.
- Edges → conditions.
- Start/terminal nodes.



Our Approach

- Learn from an example plan
- Recognize loops through loop invariants
- Use abstraction to identify similar states for determining invariants

Representation: States as Logical Structures

Dock(1)

Garage(2)

Truck(3)

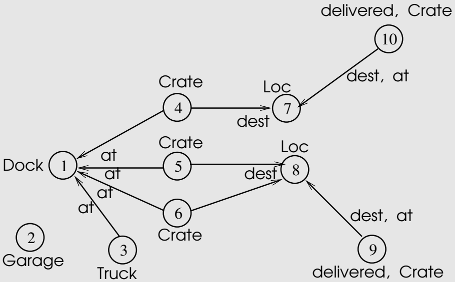
Crate(6)

Crate(10)

at(3,1)

⋮

delivered(10)



$\mathcal{V} =$
$\{Garage^1, Dock^1, Loc^1, Truck^1, Crate^1, delivered^1, at^2, in^2, dest^2\}$

$|S| = \{1, 2, \ldots 10\}$

Integrity constraints specify legal structures.

Representation: Actions

- Precondition: formula in FO(TC).
- Action operators = structure transformers

  Predicate updates
- $p'(\bar{x}) = (\neg p(\bar{x}) \wedge \Delta_p^+(\bar{x})) \quad \vee \quad (p(\bar{x}) \wedge \neg \Delta_p^-(\bar{x}))$

mv(A,B):

$$topmost'(x) = (\neg topmost(x) \wedge on(A,x)) \quad \vee \quad (topmost(x) \wedge x \neq B).$$

Review: Need for Abstraction

Idea: collapse similar states together.

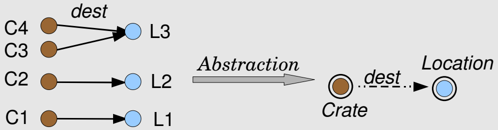
- Makes identifying invariants (recurring properties) easy.
- Use an abstraction mechanism.

We use an abstraction scheme from static analysis.

Abstraction Using 3-Valued Logic

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

- **Abstraction predicates**: chosen unary predicates.
- Values of all abstraction predicates on an element define its **role**.
- Collapse elements of the same role into **summary elements**.
- Relations involving summary elements may become **indefinite**.



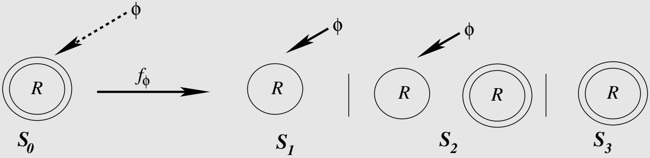
States from infinitely many instances $\mapsto$ finite set of abstract states

Precision in Action Updates

Predicate update formula:

$$p'(\bar{x}) = (\neg p(\bar{x}) \wedge \Delta_p^+) \quad \vee \quad (p(\bar{x}) \wedge \neg\Delta_p^-)$$

- TVLA's *focus+coerce* operations: make structure precise wrt a user defined formula (automatically determined in our approach).



ϕ constrained to be unique.

Use this for sensing actions too.

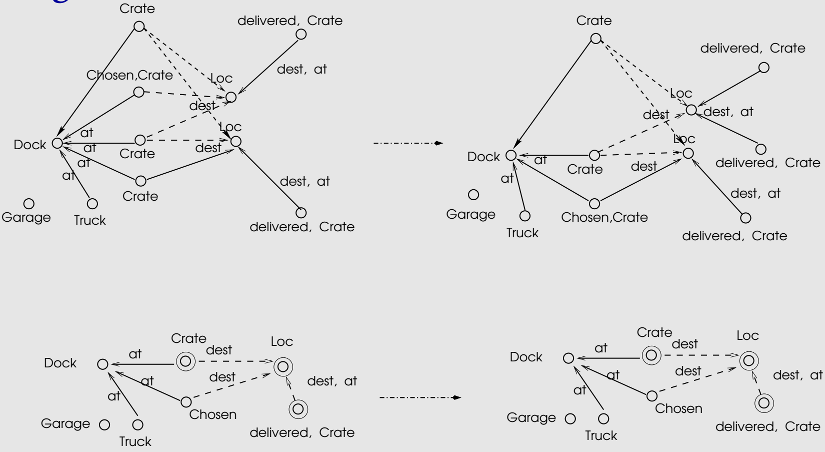
Learning Generalized Plans

We recognize loop invariants by tracing example plans in the abstract state space.

Algorithm for Learning Generalized Plans

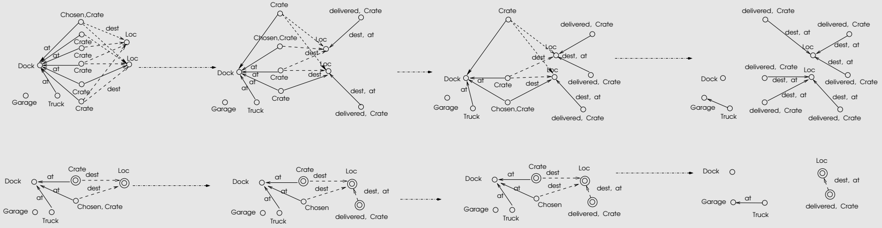
- Change action arguments to their roles in the example plan.
- Apply resulting plan to abstraction of the given start state.
- Find loops in the resulting state and action sequence.

Tracing



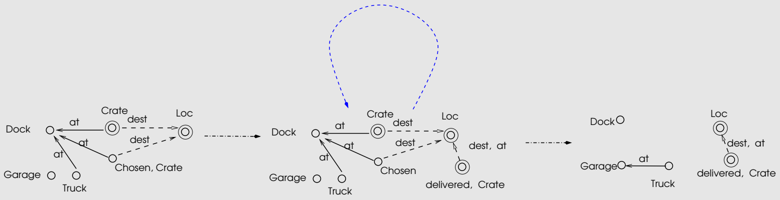
= findDest(); Load(); setTarget(destLoc); Drive(); Unload(); Choose(Crate)

Tracing



= findDest(); Load(); setTarget(destLoc); Drive(); Unload(); Choose(Crate)

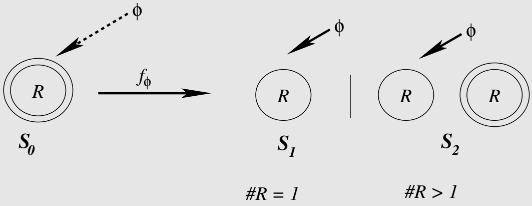
Tracing


Dock
at
Crate
dest
Loc
at
dest
at
Chosen, Crate
Garage
Truck
Dock
at
Crate
dest
Loc
at
dest
dest, at
at
Chosen
Garage
Truck
delivered, Crate
Dock
Loc
dest, at
Garage
at
Truck
delivered, Crate

Finding Preconditions

In generalized planning, correctness $\equiv$ applicability.

- **Classify** branches on the basis of *role counts*; **propagate** these counts backwards.
- Need for doing this constrains predicate update formulas.



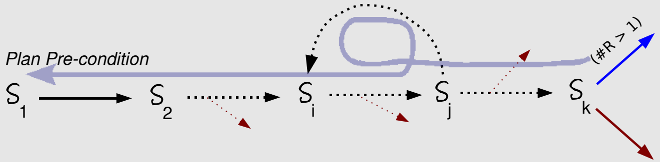
ϕ constrained to be unique and satisfiable

Finding Preconditions

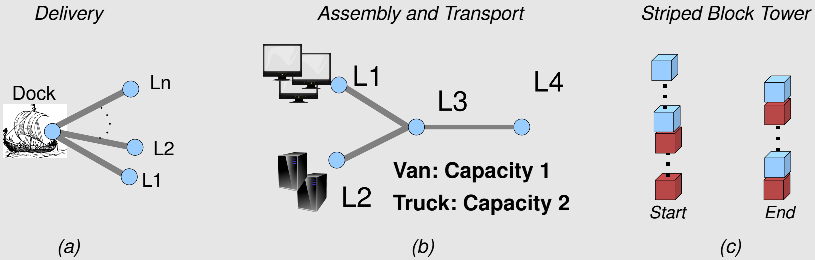
In generalized planning, correctness $\equiv$ applicability.

- **Classify** branches on the basis of *role counts*; **propagate** these counts backwards.
- Need for doing this constrains predicate update formulas.

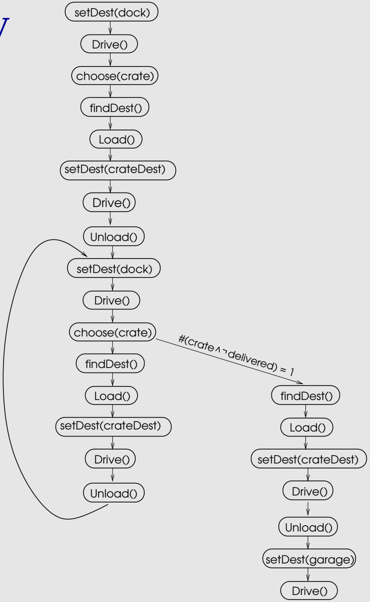
ϕ constrained to be unique and satisfiable



Problem Domains

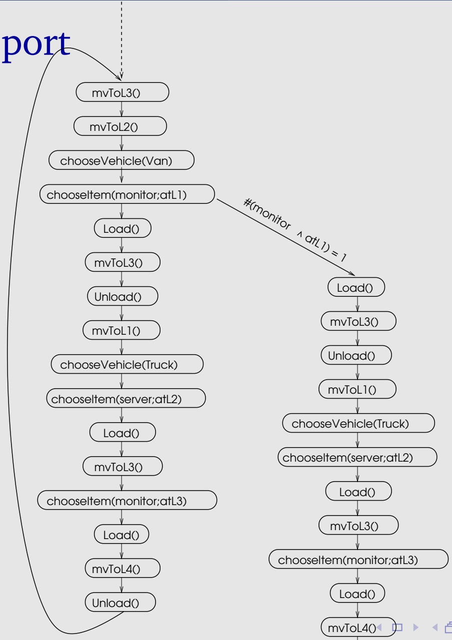


Results: Delivery

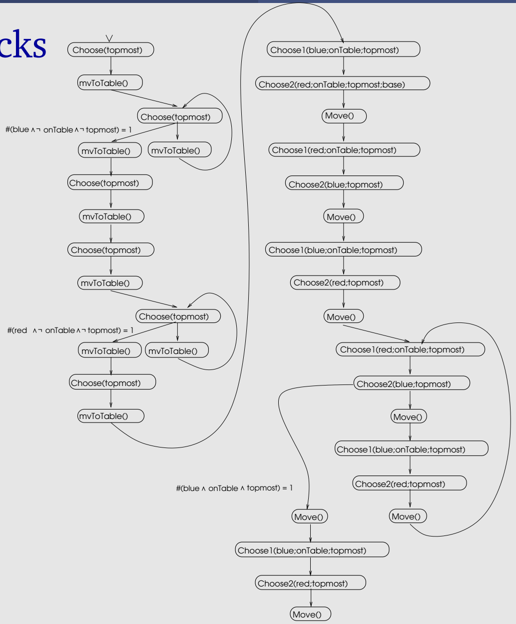


Learned plan for unit delivery

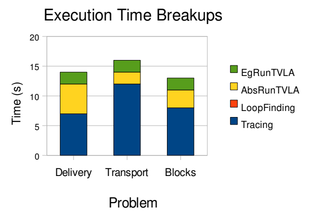
Results: Transport

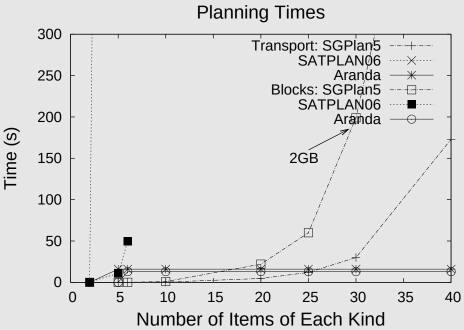


Results: Blocks



Results: Running Times





Conclusions

- Novel algorithm for generalizing plans and finding loops.
- Identified a class of domains where our methods are proven to work (extended-LL).
- No need for plan annotations/parameterization etc.

Work in Progress/Future Directions

- Plan synthesis
- Extensions beyond extended-LL domains
- Plan evaluation.

Existing Approaches

Other research along this direction

- Plan compilation: Triangle tables [Fikes et al., 1972], case based planning [Hammond, 1989]
- Explanation based learning of plans (BAGGER2) [Shavlik, 1990]
- Extracting plan templates (DISTILL) [Winner et al., 2003], planning with loops (KPLANNER) [Levesque, 2005]

Extended-LL Domains

Look like linked lists upon abstraction.

Theorem

In "extended-LL" domains, we can compute all the branch conditions and propagate them backwards to get preconditions for plans with simple loops.

We can find complete generalized plans through search in these domains!

- Defined as a set of syntactic constraints on action update formulae making sure that action updates don't require more precision than is availabe in abstract structures.
- Predicate change formulas which need focusing are role-specific, uniquely satisfiable.