Recent Advances in AI Planning: A Unified View

IJCAI-99 Tutorial

Subbarao Kambhampati

Arizona State University

rao@asu.edu

http://rakaposhi.eas.asu.edu/ijcai99-tutorial

Planning is hot...

26% of the papers in AAAI-99. 20% of papers in IJCAI-99. New People. Conferences. Workshops. Competitions. Inter-planetary explorations. Why the increased interest?

- **Significant scale-up in the last 4-5 years**
  - **Before we could synthesize about 5-6 action plans in minutes**
  - **Now, we can synthesize 100-action plans in minutes**
    - **Further scale-up with domain-specific control**
- **Significant strides in our understanding**
  - **Rich connections between planning and CSP(SAT) OR (ILP)**
    - **Vanishing separation between planning & Scheduling**
  - **New ideas for heuristic control of planners**
  - **Wide array of approaches for customizing planners with domain-specific knowledge**

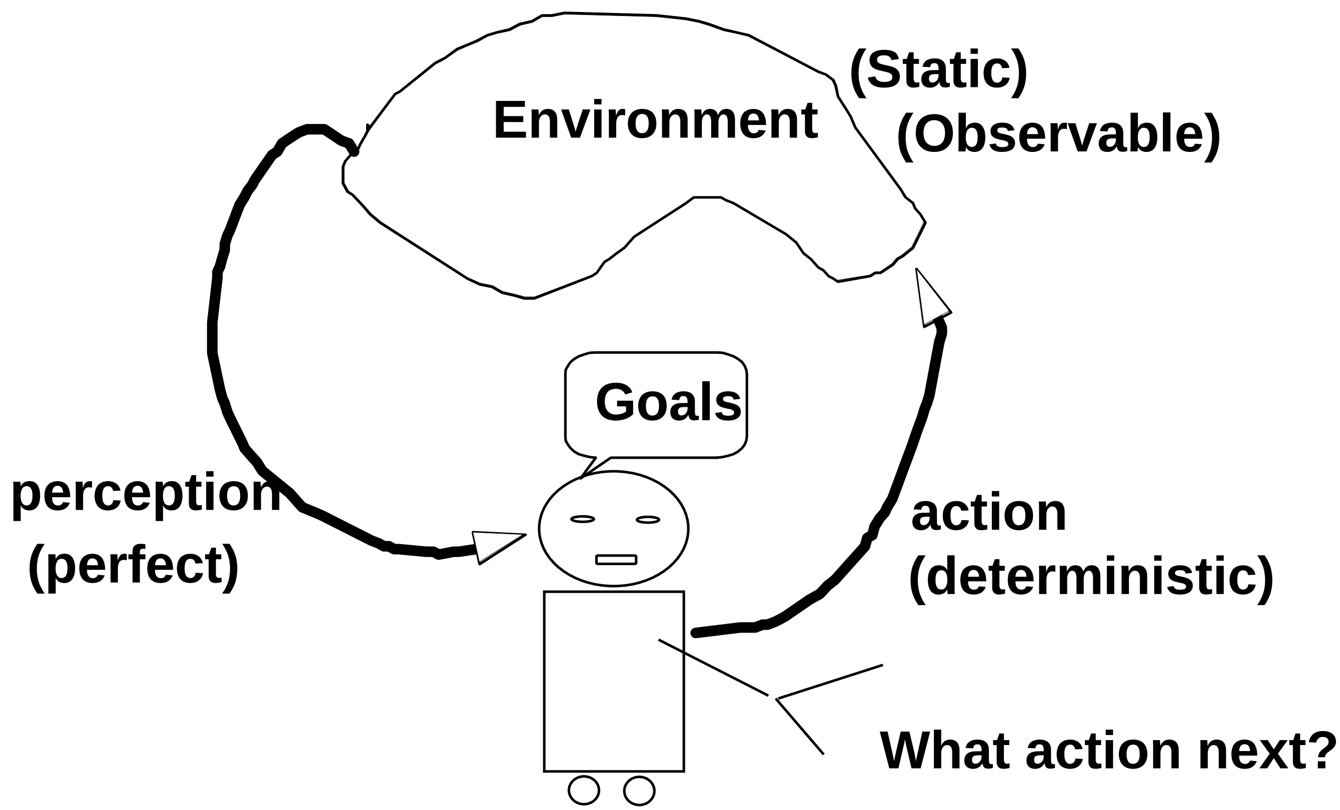
Overview

- (Classical) planning problem
  - Modeling, Proving correctness
- Refinement Planning: Formal Framework
- Conjunctive refinement planners
  - Heuristics
- Disjunctive refinement planners
  - Refinement of disjunctive plans
  - Solution extraction from disjunctive plans
    - Direct, Compiled (SAT, CSP, ILP)
- Customizing Planners
  - User-assisted Customization
  - Automated customization
- Support for non-classical worlds

Planning : The big picture

- Synthesizing goal-directed behavior
- Planning involves
  - Action selection; Handling causal dependencies
  - Action sequencing and handling resource allocation (aka SCHEDULING)
- Depending on the problem, plans can be
  - action sequences
  - or “policies” (action trees, state-action mappings etc.)

Planning & (Classical Planning)



I = initial state **G** = goal state

(prec) O_i (effects)

[I] O_i → O_j → → O_k → O_m [G]

Why care about classical Planning?

- Many domains are approximately classical
  - Stabilized environments
- It is possible to handle near-classical domains through replanning and execution monitoring
- Classical planning techniques often shed light on effective ways of handling non-classical planning worlds
  - Currently, most of the efficient techniques for handling non-classical scenarios are still based on ideas/advances in classical planning
- Classical planning poses many interesting computational challenges

The (too) many brands of classical planners

Planning as Theorem Proving (Green's planner) → **Planning as Search**

Search in the space of States (progression, regression, MEA) (STRIPS, PRODIGY, TOPI)

Search in the space of Plans (total order, partial order, protections, MTC) (Interplan,SNLP,TOCL, UCPOP,TWEAK)

Search in the space of Task networks (reduction of non-primitive tasks) (NOAH, NONLIN, O-Plan, SIPE)

Planning as (constraint) Satisfaction (Graphplan, IPP, STAN, SATPLAN, BLackBOX)

Unifying Framework

Advantages of the Unified View

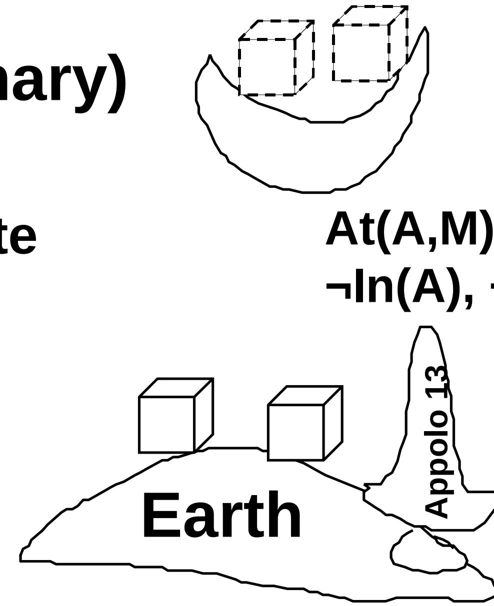
To the extent possible, this tutorial shuns brand names and reconstructs important ideas underlying those brand names in a rational fashion

- Better understanding of existing planners
  - Normalized comparisons between planners
  - Evaluation of trade-offs provided by various design choices
- Design of novel planning algorithms
  - Hybrid planners using multiple refinements
  - Explication of the connections between planning, CSP, SAT and ILP

Modeling Classical Planning: Actions, States, Correctness

Modeling Classical Planning

- ✧States are modeled in terms of (binary) state-variables
  - -- Complete initial state, partial goal state
- ✧Actions are modeled as state transformation functions
  - -- Syntax: ADL language (Pednault)
  - -- Apply(A,S) = (S \ eff(A)) + eff(A) (If Precond(A) hold in S)



At(A,M),At(B,M)
¬In(A), ¬In(B)

At(A,E), At(B,E),At(R,E)

Effects

Prec.

Action	Preconditions	Effects
$\text{Load}(o_1)$	$At(o_1,l_1), At(R,l_1)$	$In(o_1)$
$\text{Unload}(o_1)$	$In(o_1)$	$\neg In(o_1)$
$\text{Fly}()$	$At(R,E)$	$At(R,M), \neg At(R,E)$; $\forall_x In(x) \Rightarrow At(x,M)$ & $\neg At(x, E)$

Some notes on action representation

- STRIPS Assumption: Actions must specify all the state variables whose values they change...
- No disjunction allowed in effects
  - Conditional effects are NOT disjunctive
    - » (antecedent refers to the previous state & consequent refers to the next state)
- Quantification is over finite universes
  - essentially syntactic sugaring
- All actions can be compiled down to a canonical representation where preconditions and effects are propositional
  - Exponential blow-up may occur (e.g removing conditional effects)
    - » *We will assume the canonical representation*

Action A

Eff: If P then R
If Q then W

Action A1
Prec: P, Q
Eff: R, W

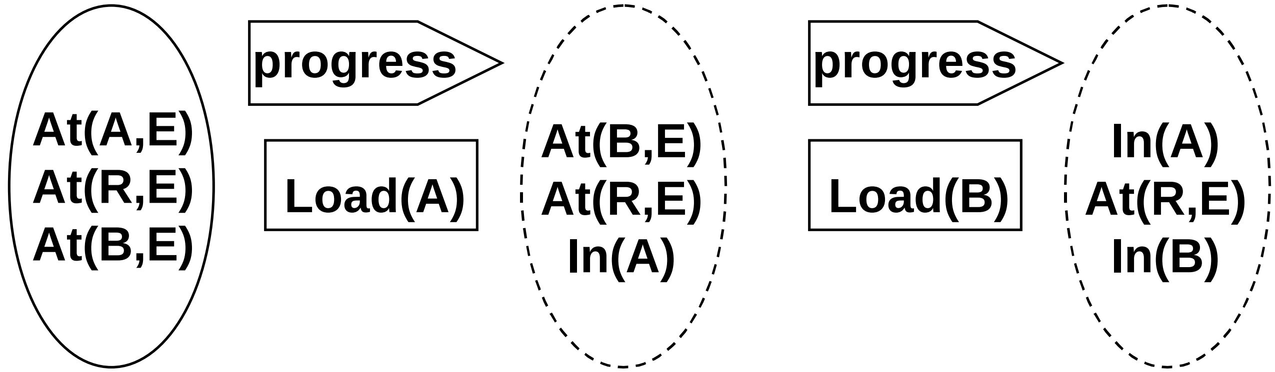
Action A2
Prec: P, ~Q
Eff: R, ~W

Action A3
Prec: ~P, Q
Eff: ~R, W

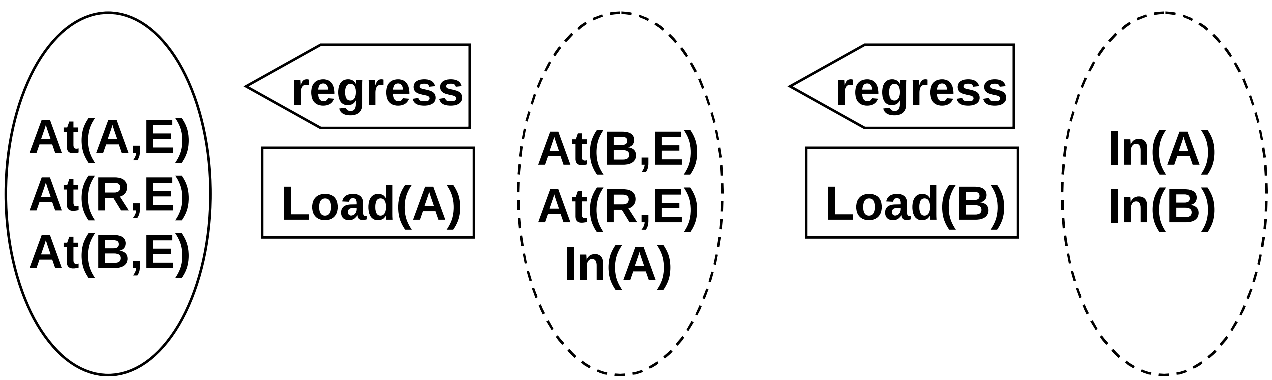
Action A4
Prec: ~P,~Q
Eff:

Checking correctness of a plan: The State-based approaches

- Progression Proof: Progress the initial state over the action sequence, and see if the goals are present in the result



- Regression Proof: Regress the goal state over the action sequence, and see if the initial state subsumes the result



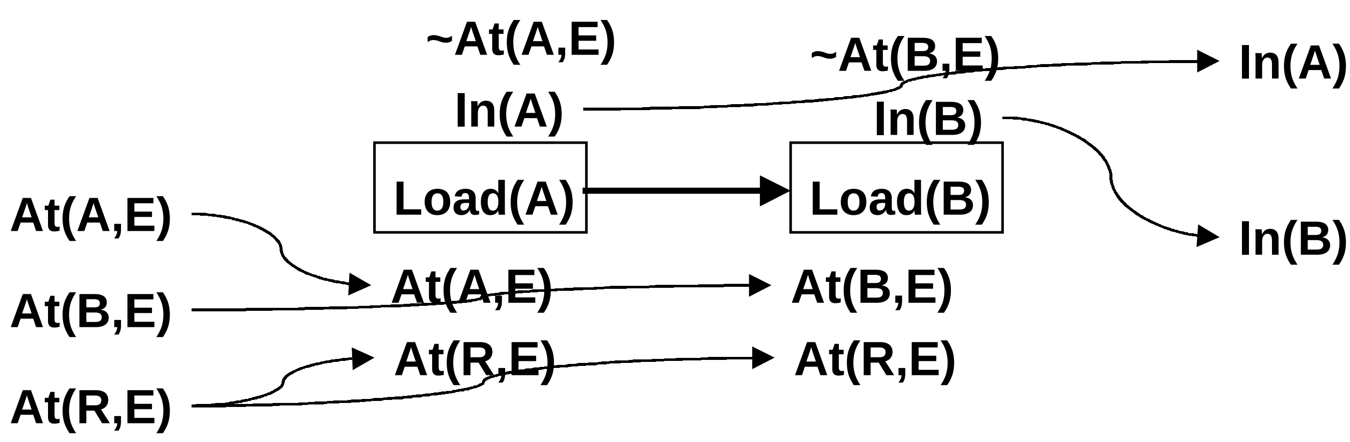
Checking correctness of a plan: The Causal Approach

Contd..

- Causal Proof: Check if each of the goals and preconditions of the action are
  - » "established" : There is a preceding step that gives it
  - » "declobbered": No possibly intervening step deletes it
    - Or for every preceding step that deletes it, there exists another step that precedes the conditions and follows the deleter adds it back.

Causal proof is

- "local" (checks correctness one condition at a time)
- "state-less" (does not need to know the states preceding actions)
- "incremental" with respect to action insertion



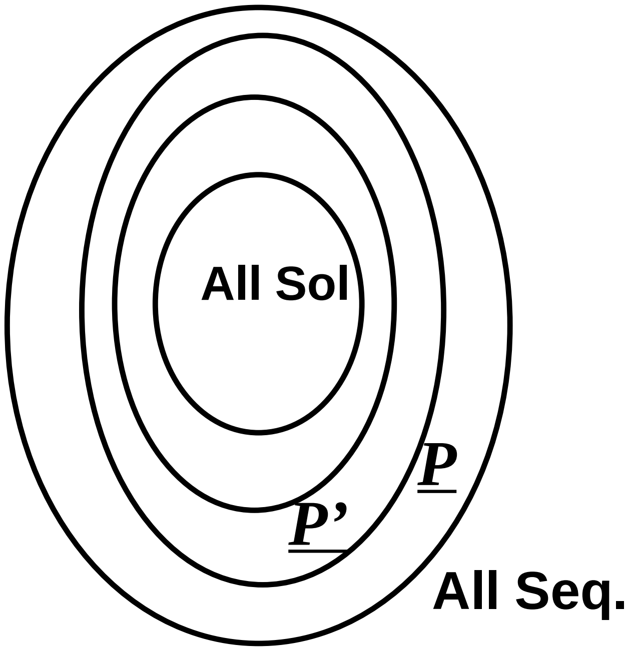
The Refinement Planning Framework:

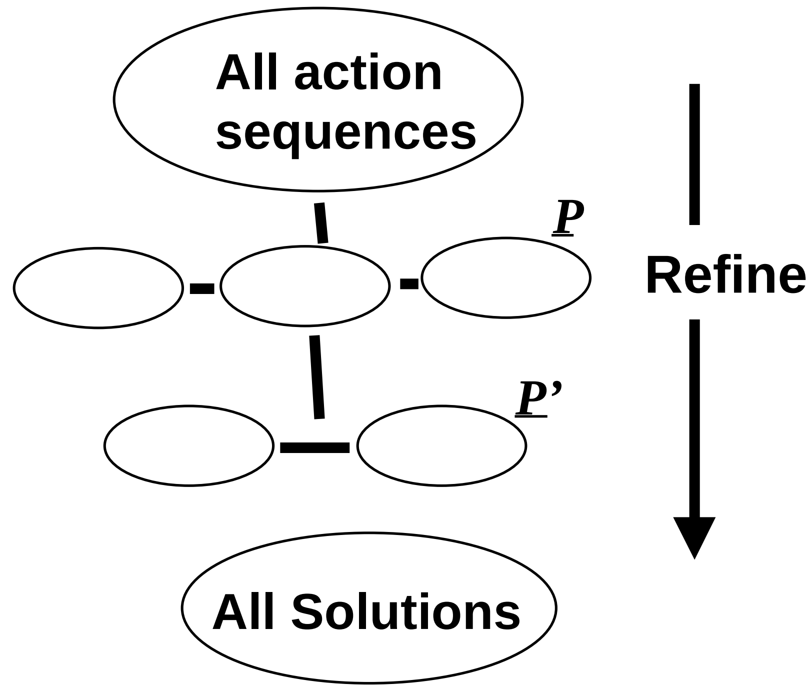
1. Syntax & Semantics of partial plans

2. Refinement strategies & their properties

3. The generic Refinement planning template

Refinement Planning:Overview





Refinements

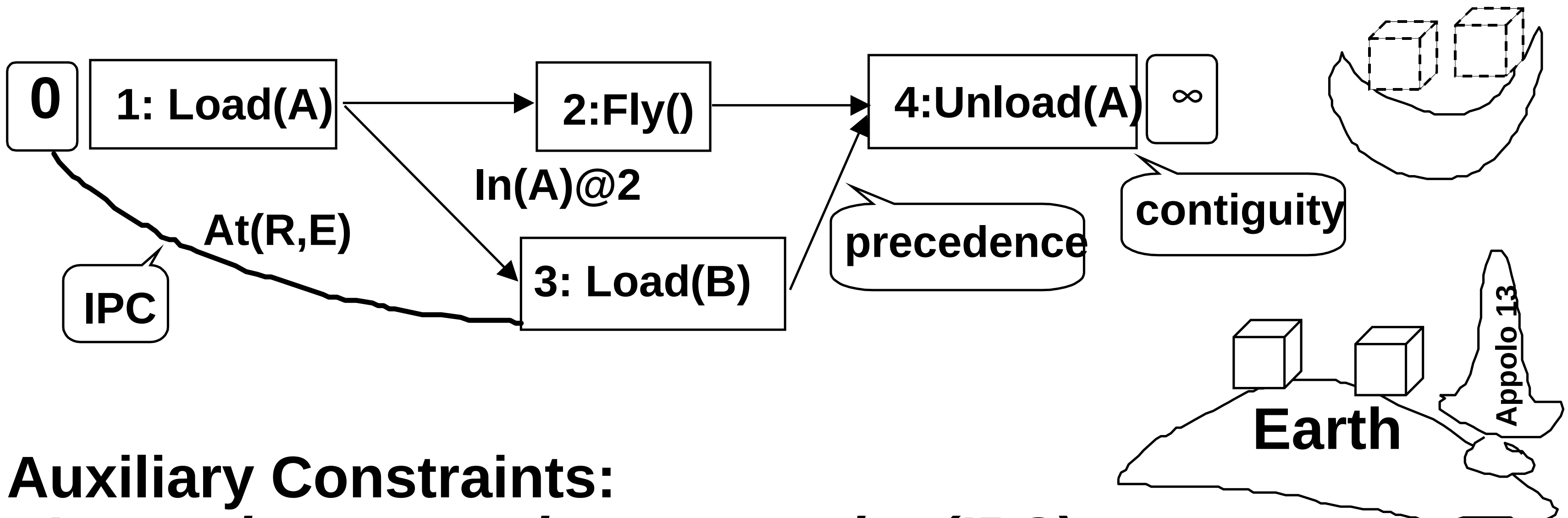
Partial plans

Narrowing *sets of action sequences* to progress towards solutions

Remove non-solutions

Partial Plans: Syntax

Partial plan = ⟨ Steps, Orderings, Aux. Constraints ⟩



Auxiliary Constraints:

Interval preservation constraint (IPC) **‹ s_1 , p , s_2 ›**

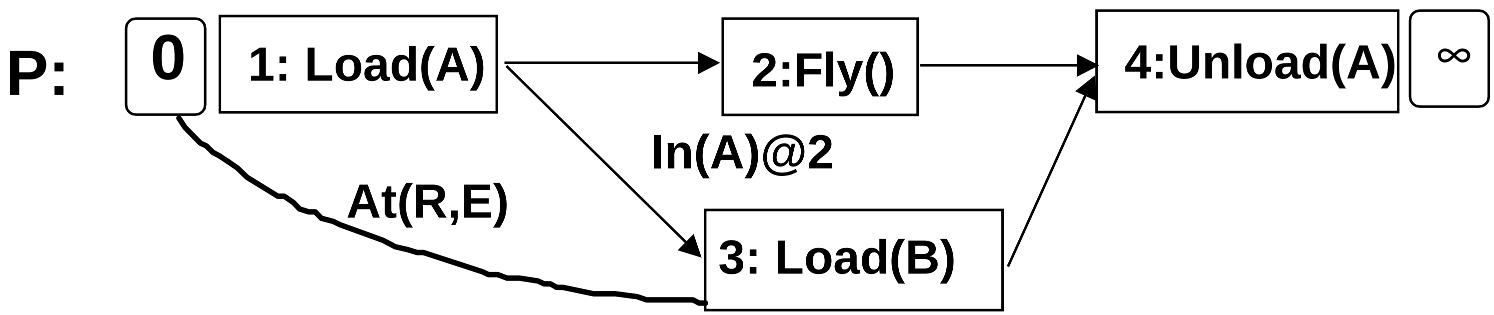
p must be preserved between s_1 and s_2

Point truth Constraint (PTC) **p@s**

p must hold in the state before s

Partial Plans: Semantics

Candidate is any action sequence that
- **-- contains actions corresponding to all the steps,**
- **-- satisfies all the ordering and auxiliary constraints**



Candidates ($\in$ «P»)

[Load(A),Load(B),Fly(),Unload(A)]

Minimal candidate. Corresponds to safe linearization [01324 ∞]

[Load(A),Load(B),Fly(), <u>Unload(B)</u>,Unload(A)]

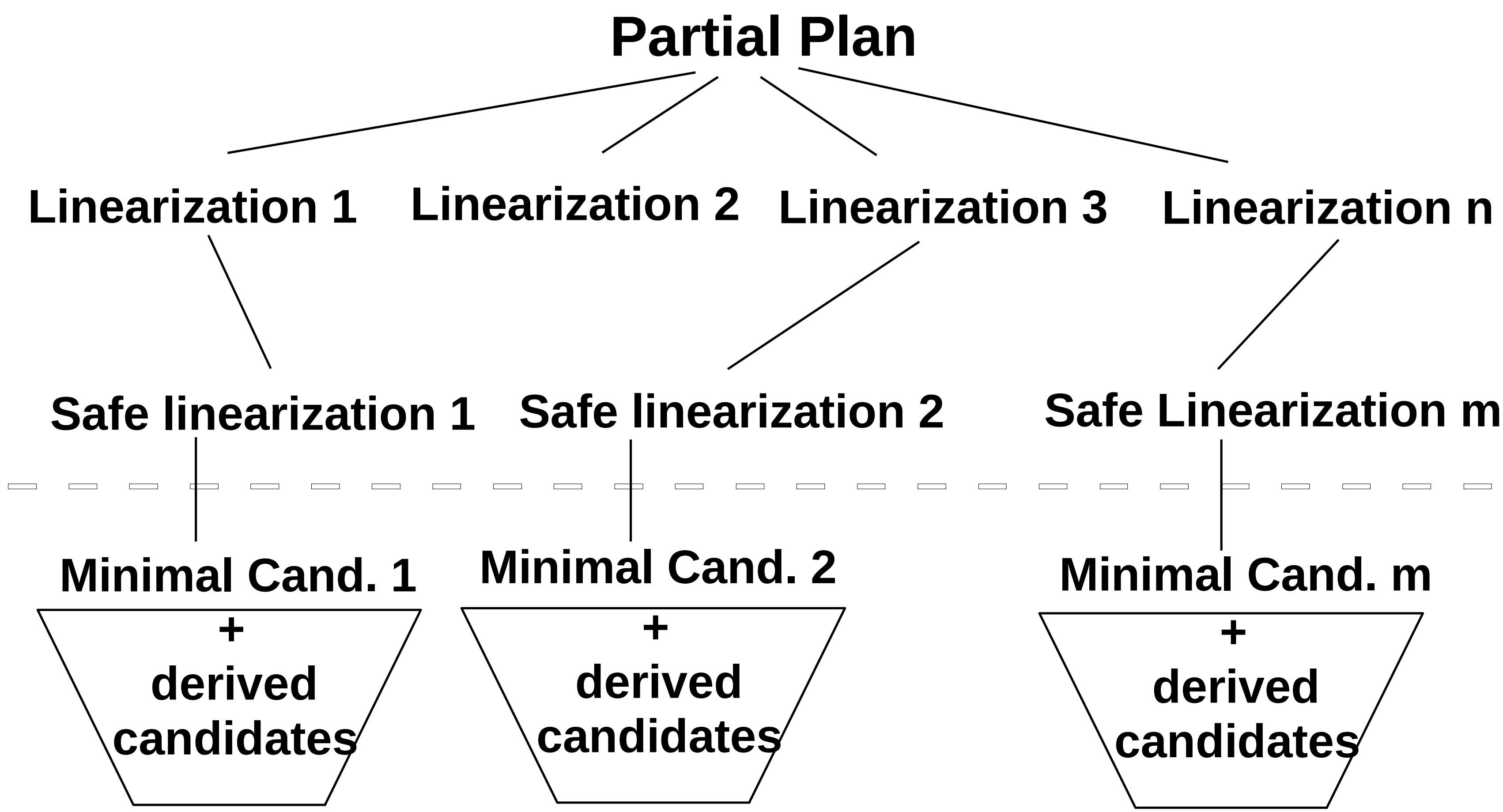
Non-Candidates ($\notin$ «P»)

[Load(A),Fly(),Load(B),Unload(B)]

Corresponds to unsafe linearization [01234 ∞]

[Load(A),<u>Fly()</u>,Load(B), Fly(),Unload(A)]

Linking Syntax and Semantics



Refinements
- **Reduce candidate set size**
- **Increase length of minimal candidates**

Refinement (pruning) Strategies

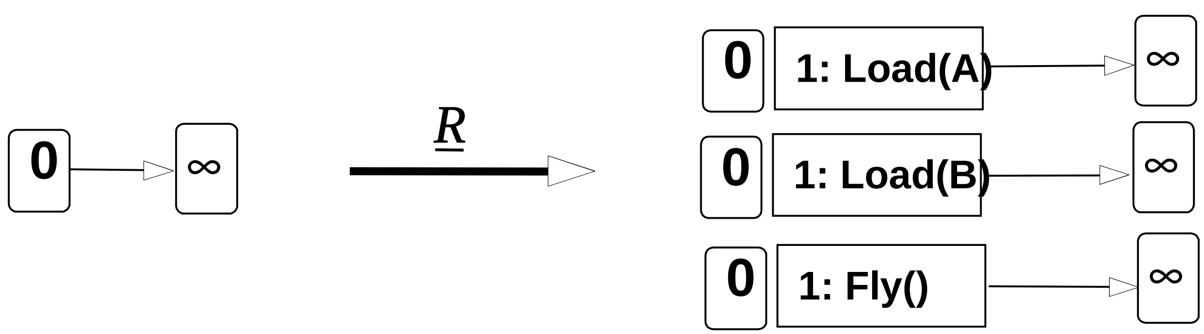
"Canned" inference procedures

actions & their inter-relations

-- Prune by propagating the consequences of domain theory and meta-theory of planning onto the partial plan

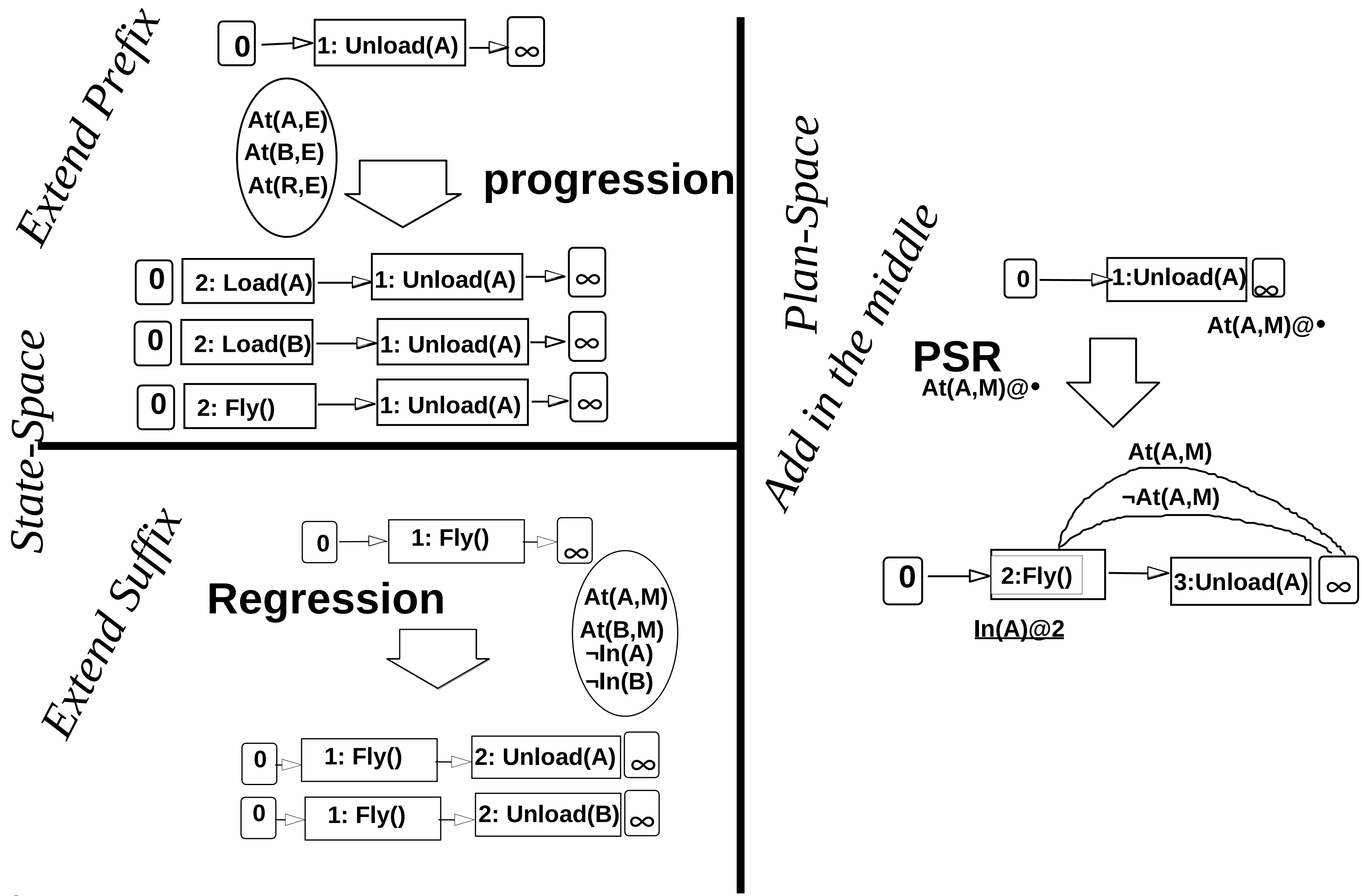
✧A refinement strategy **R** : ***P*** ↣ ***P'*** («***P'*** » a subset of «***P*** »)

- –**R** is complete if «***P'*** » contains all the solutions of «***P*** »
- –**R** is monotonic if «***P'*** » has longer minimal candidates than «***P*** »
- –**R** is progressive if «***P'*** » is a proper subset of «***P*** »
- –**R** is systematic if components of ***P'*** don't share candidates

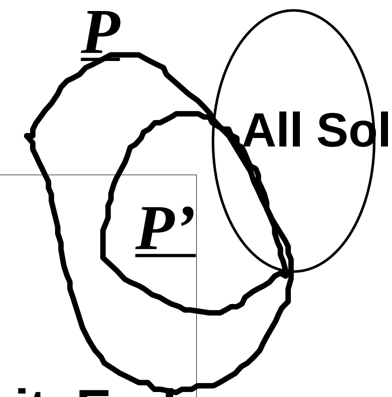


§ A *plan set* ***P*** is a set of partial plans $\{P_1, P_2 \ldots P_m\}$

Existing Refinement Strategies



The Refinement Planning Template



Refineplan(*P* : Plan set)

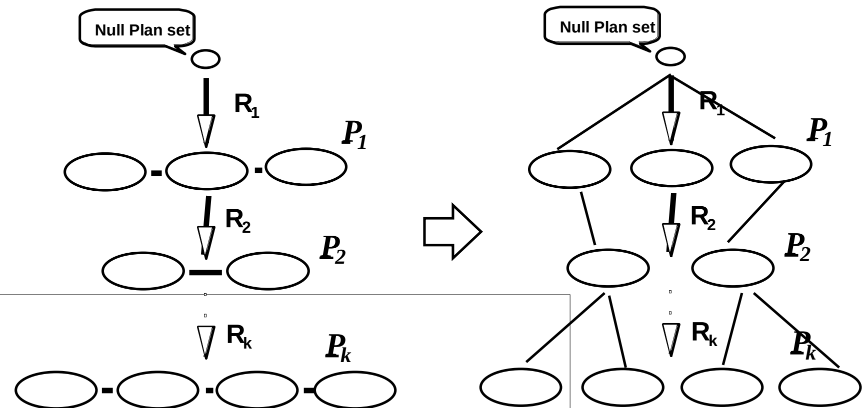
0*. If «*P* » is empty, Fail.
1. If a minimal candidate of *P* is a solution, return it. End
2. Select a refinement strategy **R**
   Apply **R** to *P* to get a new plan set *P'*
3. Call Refine(*P'*)

-- Termination ensured if **R** is complete and monotonic
-- Solution check done using one of the proofs of correctness

Issues:

1. Representation of plan sets (Conjunctive vs. Disjunctive)
2. Search vs. solution extraction
3. Affinity between refinement and proof used for solution check

A flexible Split&Prune search for refinement planning



Refineplan(*P* : Plan)

0*. If «*P* » is empty, Fail.
1. If a minimal candidate of *P* is a solution, terminate.
2. Select a refinement strategy R . Appply R to *P* to get a new plan set *P'*
3. Split *P'* into k plansets
4. Non-deterministically select one of the plansets P'_i Call Refine(P'_i)

Two classes of refinement planners

Then

Conjunctive planners

- Search in the space of conjunctive partial plans
  - Disjunction split into the search space
    - » search guidance is nontrivial
  - Solution extraction is trivial
- Examples:
  - STRIPS & Prodigy
  - SNLP & UCPOP
  - NONLIN & SIPE
  - UNPOP & HSP

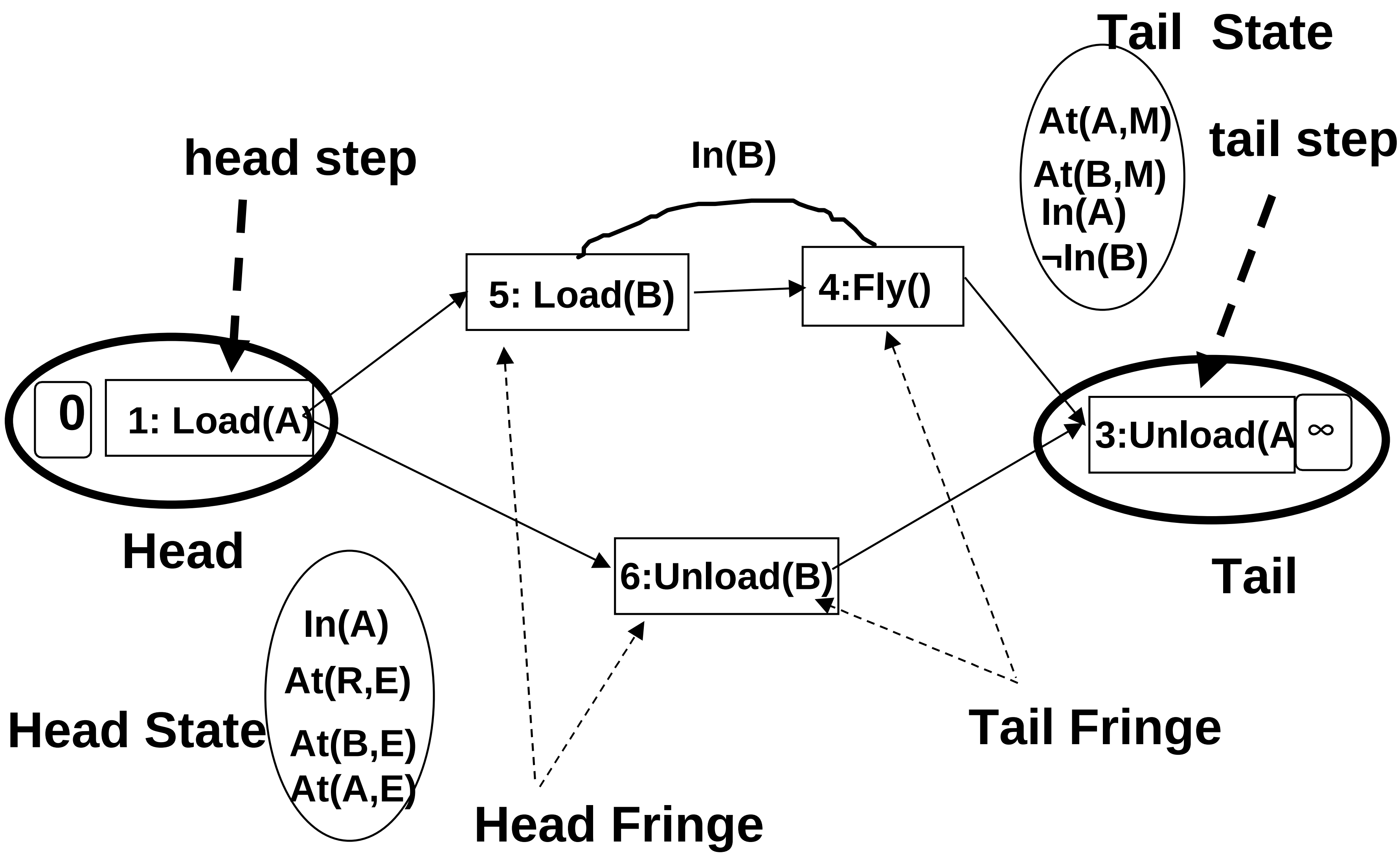
Now

Disjunctive planners

- Search in the space of disjunctive partial plans
  - Disjunction handled explicitly
  - Solution extraction is non-trivial
    - » CSP/SAT/ILP methods
- Examples:
  - Graphplan
  - SATPLAN

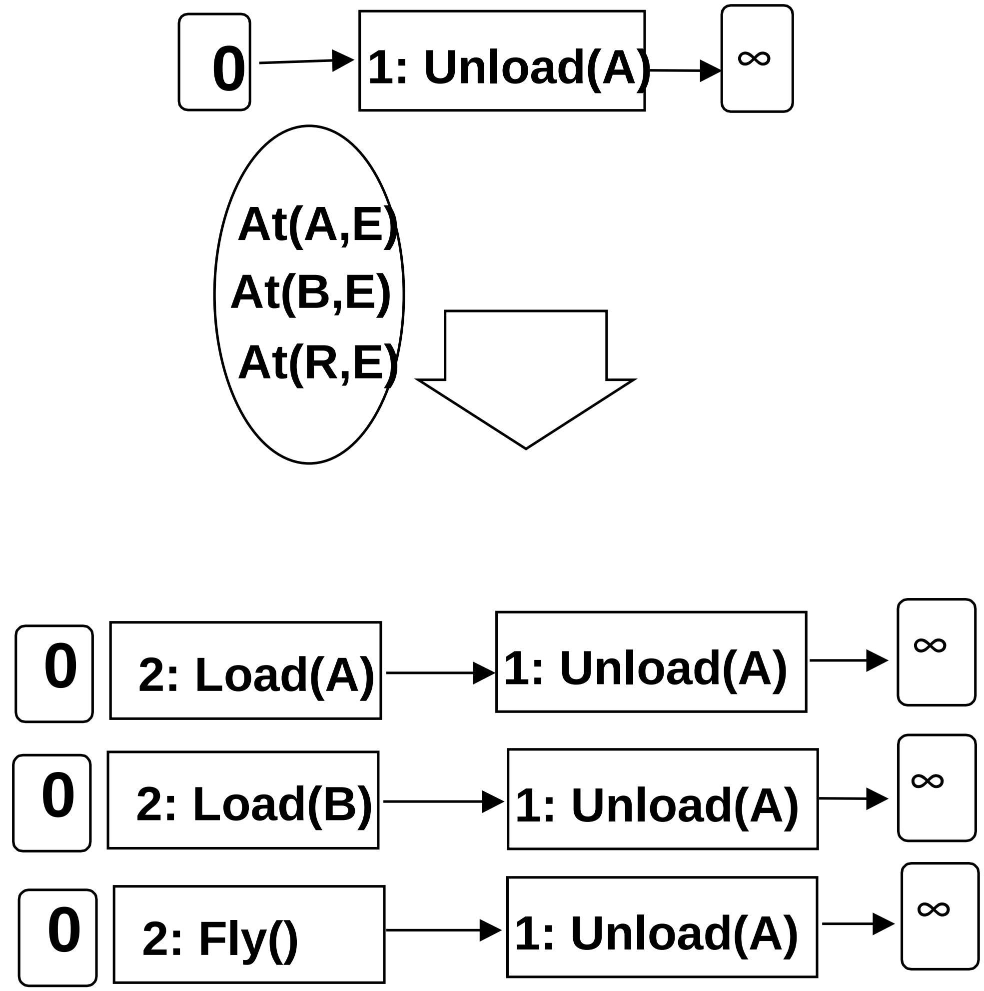
CONJUNCTIVE REFINEMENT PLANNING

Plan structure nomenclature



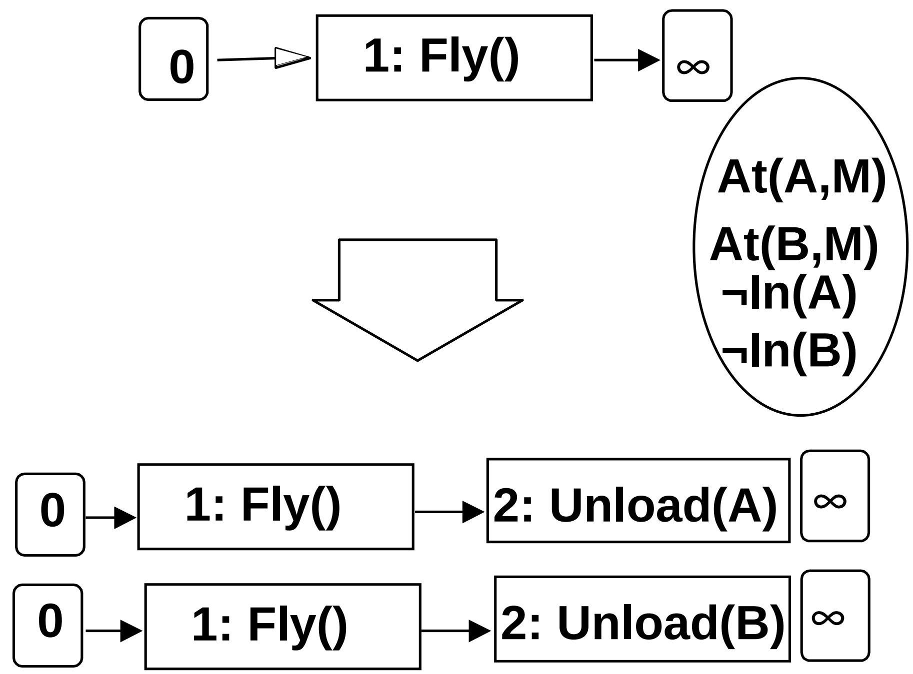
Forward State-space Refinement

- **Grow plan prefix by adding applicable actions**
  - **Complete, Monotonic**
    - » ***consideration of all executable prefixes***
  - **Progressive**
    - » ***elimination of unexecutable prefixes***
  - **Systematic**
    - » ***each component has a different prefix***
  - **Completely specified state**
    - » **Easier to control?**



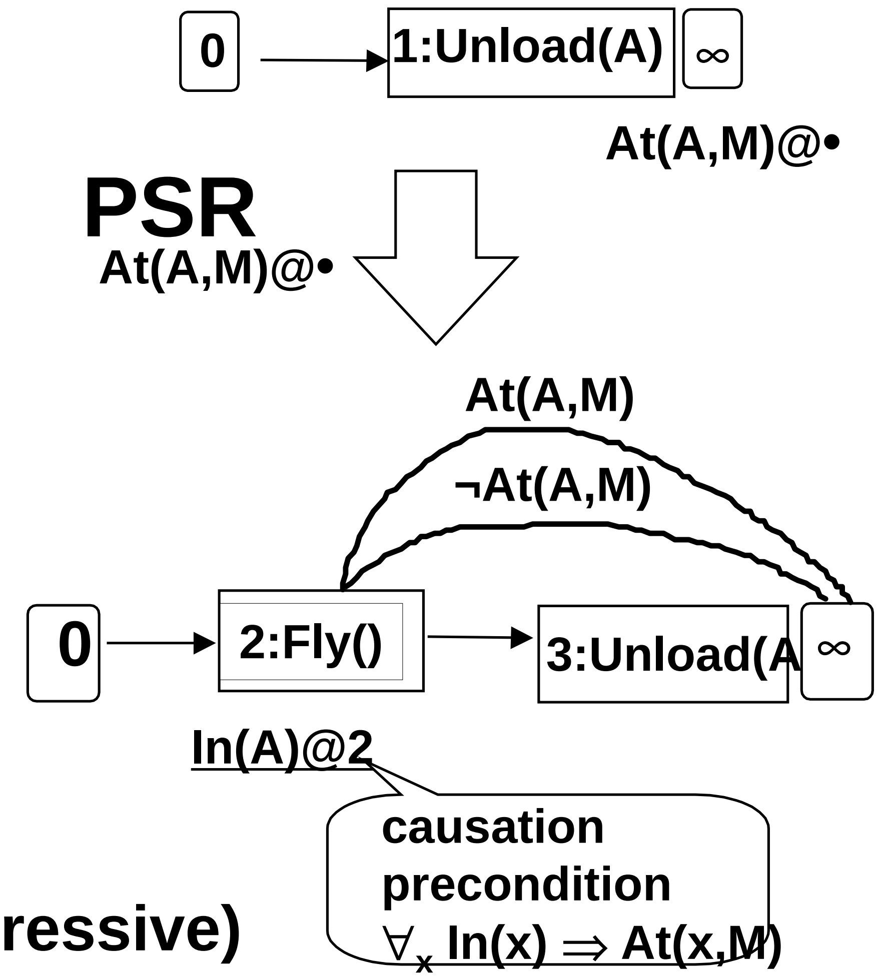
Backward State-space Refinement

- Grow plan suffix by adding relevant actions
  - Complete, Monotonic
    - » *consideration of all relevant suffixes*
  - Progressive
    - » *elimination of irrelevant suffixes*
  - Systematic
    - » *each component has a different suffix*
  - ✚ Goal directed
    - » Lower branching



Plan-space Refinement

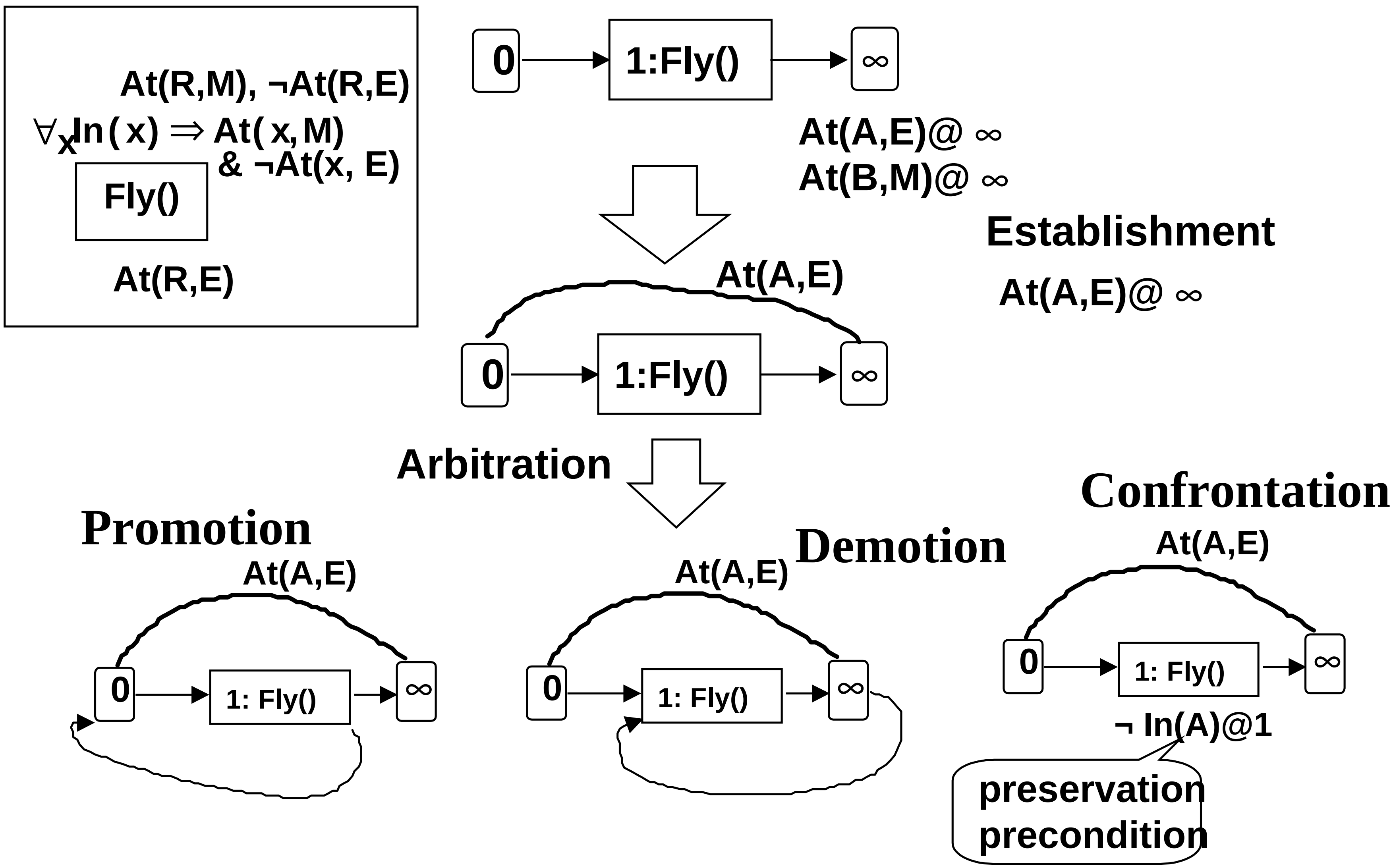
Goal selection:
  Select a precondition
Establishment:
  Select a step (new or existing) and make it give the condition
De-clobbering:
  Force intervening steps to preserve the condition
Book-keeping: (Optional)
  Add IPCs to preserve the establishment
  $\Rightarrow$ Systematicity



(PSR is complete, and progressive)

(Sacerdoti, 1972; Pednault, 1988; McAllester, 1991)

Plan-space Refinement: Example 2



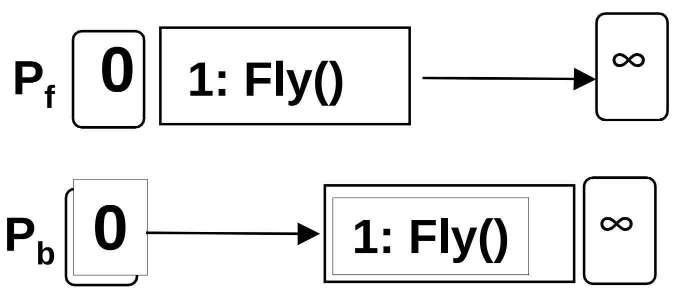
Position, Relevance and Commitment

The Tastes great/
Less Filling debate

FSR and BSR must commit to both position and relevance of actions

+ Gives state information

- Leads to premature commitment



Plan-space refinement (PSR) avoids constraining position

+ Reduces commitment

- Increases plan-handling costs

Generating Conjunctive Refinement Planners

Refineplan (*P* : Plan)

0*. If «*P* » is empty, Fail.
1. If a minimal candidate of *P* is a solution, terminate.
2. Select a refinement strategy R .
Appply R to *P* to get a set of new plans *P1…Pj*
Add all plans to the search queue.
4. Non-deterministically select one of the plans P_i
Call Refine(P_i)

Keep all linearizations safe
(Tractability refinements)

Select the proof strategy so it becomes
incremental with respect to the refinement

Stick to a single refinement strategy

Issues in instantiating Refineplan

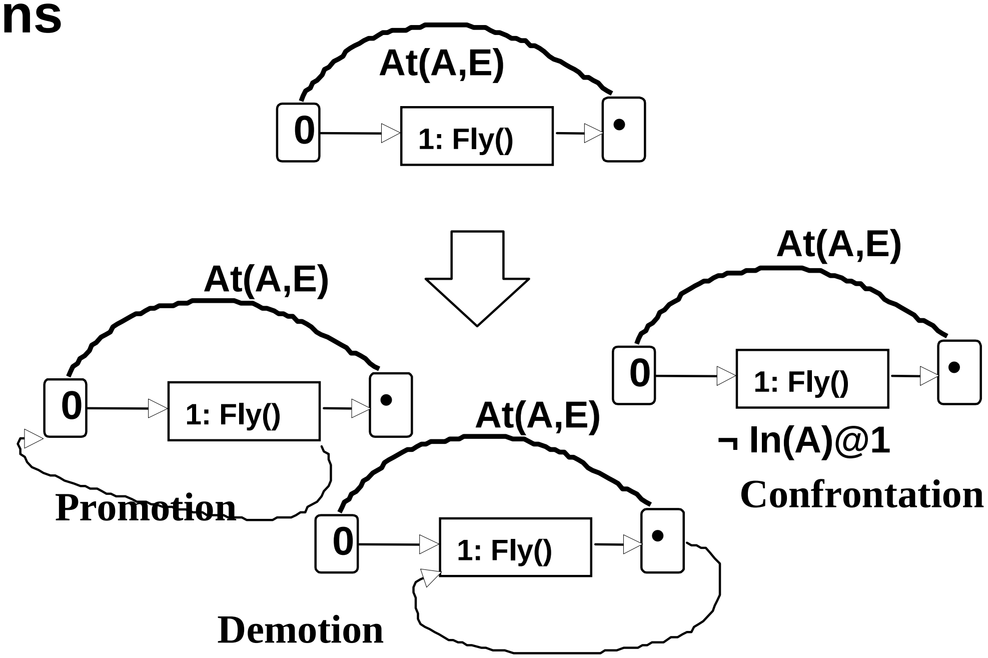
- **Although a planner can use multiple different refinements, most implementations stick to a single refinement**
- **Although refinement can be used along with any type of correctness check, there is affinity between specific refinements and proof technqiues (support finite differencing)**
  - **FSR and Progression based proof**
  - **BSR and Regression based proof**
  - **PSR and Causal proof**
- **Although it is enough to check if *any one* of the safe linearizations are solutions, most planners refine a partial plan until all its linearizations are safe**
  - **Tractability refinements (pre-order, pre-satisfy)**

With these changes, an instantiated and simplified planner may "*look*" considerably different from the general template

Tractability Refinements

Aim: Make it easy to generate minimal candidates

- **Reduce number of linearizations**
  - **Pre-ordering**
  - **Pre-positioning**
- **Make all linearizations safe**
  - **Pre-satisfaction**
    - » **Resolve threats to auxiliary constraints**
- **Reduce uncertainity in action identity**
  - **Pre-reduction**
    - » **Replace a non-primitive action with its reductions**



Case Study: UCPOP

Refineplan (***P*** : Plan)

0*. If P is order inconsistent, FAIL.

1. If no open conditions and no unsafe IPCs, SUCCESS.

2. Generate new plans using either 2' or 2''
   Add the plans to the search queue

   2'. Remove an open condition c@s in P.

   2.1. For each step s' in P that gives c, make a new plan
   P' = P + (s' < s) + IPC s'-c-s

   2.2. For each action A in the domain that gives c, make a new plan P' = P + sn:A + (sn <s) + IPC sn-c-s.

   2.2.1. For each precondition c' of A, add c'@sn to the list of open conditions of P'.

   2.2.2. For each IPC s'-p-s'', if sn deletes p, add [s'-p-s''; sn] to the list of unsafe IPCs of P'.

   2''. Remove an unsafe IPC [s'-p-s''; s'''] from P.
   Make two plans: P' = P + s'''< s' P'' = P + s'' < s'''

3. Non-deterministically select one of the plans P_I from the search queue and Call Refine($\underline{P}_i$)

Incrmental Soln. check

Finite-differencing structures

Tractability refinement

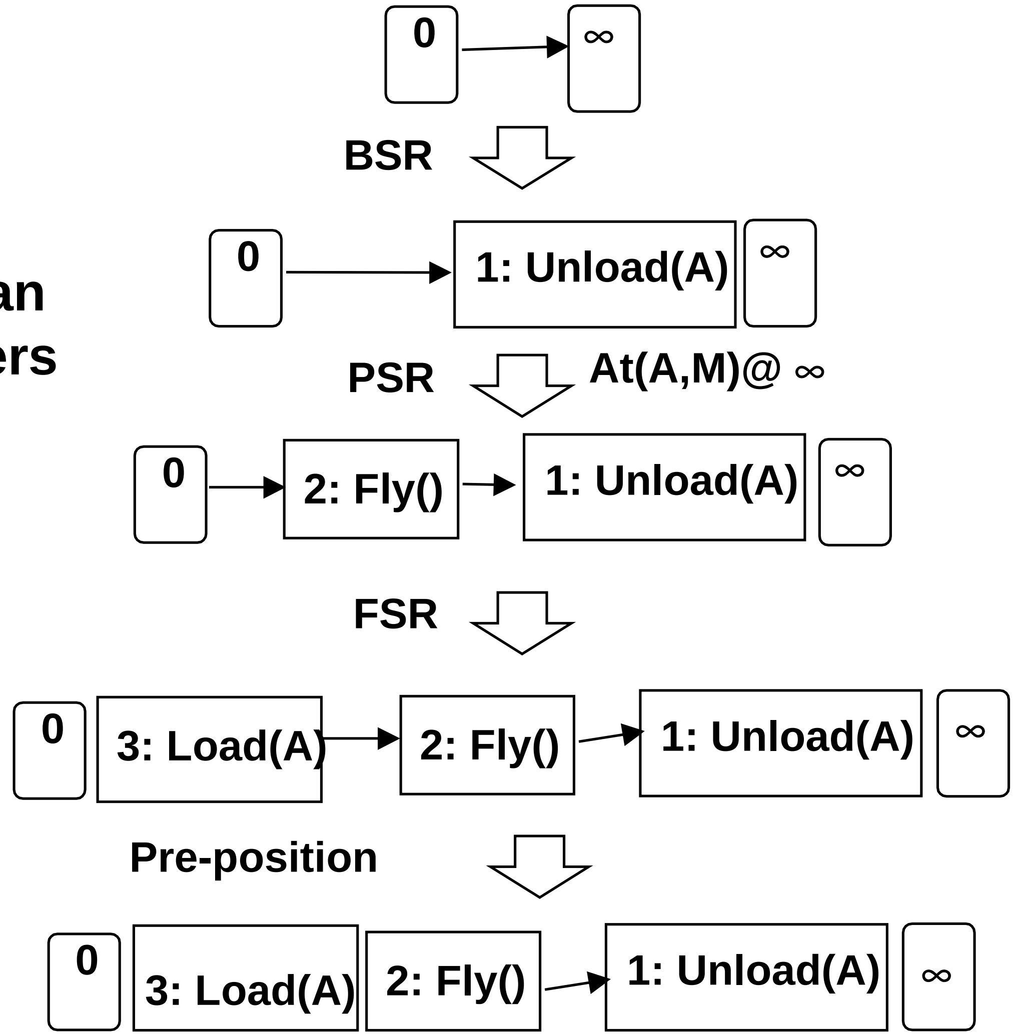
Many variations on the theme..

Planner	*Termination*	*Goal Sel.*	*Bookkeeping*	*Tractability refinements*
TWEAK	MTC	MTC	*-none-*	*-none-*
UA	MTC	MTC	*-none-*	pre-order
SNLP / UCPOP	Causal proof	arbitrary	Contrib. Prot	pre-satisfaction
TOCL	Causal proof	arbitrary	Contrib. Prot.	pre-order
McNonlin/ Pedestal	Causal proof	arbitrary	Interval Prot.	pre-satisfaction
SNLP-UA	MTC	MTC	Contrib. Prot.	unambig. ord.

(Kambhampati, Knoblock & Yang, 1995)

Interleaving Refinements

- Combine different refinements opportunistically
  - Can be more efficient than single-refinement planners
  - Refinement selection criteria?
    - » # Components produced
    - » "Progress" made



(Kambhampati & Srivastava, 1995)

Effective Heuristics for conjunctive planners

- State-space planners can be focused with greedy regression graphs (see below)
- Plan-space planners are focused by “flaw selection” strategies
  - Select the flaw (open condition, unsafe IPC) that has the fewest number of resolution possibilities
    - (Fail first heuristic)
  - Maintain the live resolution possibilities for each flaw (Descartes)
    - (Forward Checking)

Greedy Regression Graphs

Problem: Estimate the length of the plan needed to go from the head-state of the current partial plan to the goal state.

Solution:
--Relax the problem by assuming that all subgoals are independent (ignore +ve / -ve interactions between actions)
--Solve the relaxed problem and use the length of the solution as part of the heuristic

Properties: The heuristic is neither a lower bound (-ve interactions) nor an upper-bound (+ve interactions).
--leads to inoptimal solutions (in terms of plan length)
>>Possible to reduce inoptimality by considering interactions

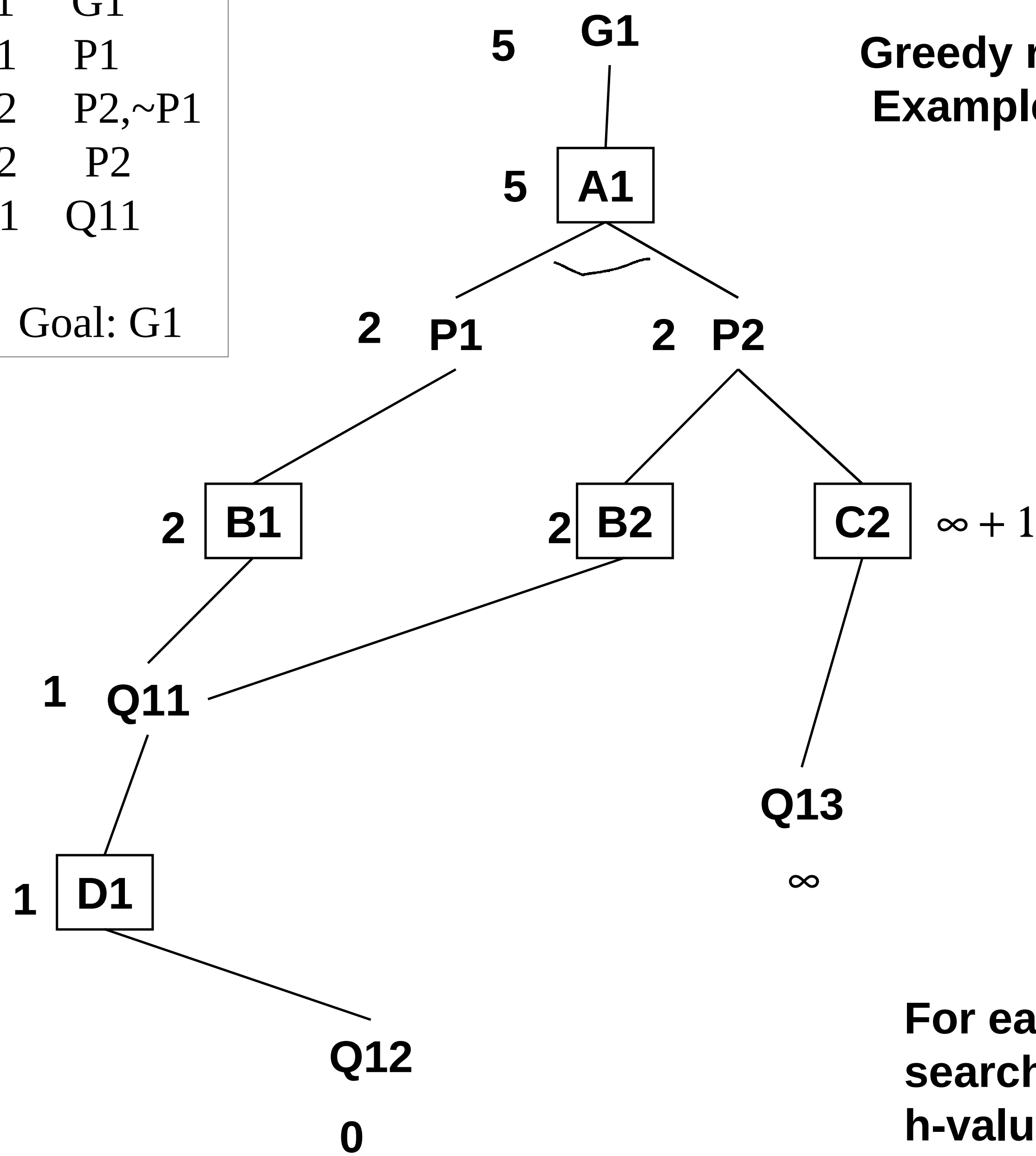
The heuristic can also be made to work in the reverse direction

Planners: UNPOP (McDermott); HSP (Bonet & Geffner)

P1,P2	A1	G1
Q11	B1	P1
Q11	B2	P2,~P1
Q13	C2	P2
Q12	D1	Q11

Init: Q12 Goal: G1

Greedy regression Graph: Example



+ve as well as -ve interactions between P1 and P2 are ignored….

For each partial plan in the search queue, estimate its h-value using this procedure

Some implemented conjunctive planners

	Refinement	Heuristics
UCPOP, SNLP [Weld et. al.]	Plan-space	Fail-first Flaw selection
UNPOP [McDermott] HSP [Geffner & Bonet]	Forward state space	Greedy regression
Descartes [Joslin & Pollack] UCPOP-D [Kambhampati & Yang]	Plan-space	Forward checking + Fail-first flaw selection
Prodigy [Carbonell et. al.]	Forward state space	Means-ends analysis

Conjunctive planners: The State of the Art

- Vanilla state-space (FSR/BSR) planners were known to be less efficient than vanilla plan-space planners
  - Several research efforts concentrated on extending plan-space approaches to non-classical scenarios
- Forward state-space planners using greedy regression heuristic are competitive with the best available planners
  - At this time, there do not seem to be comparably effective heuristics for plan-space planners.
- Plan-space planners still provide better support for incremental planning (replanning, reuse and plan modification)

DISJUNCTIVE REFINEMENT PLANNING

Disjunctive Planning

- Idea: Consider Partial plans with disjunctive step, ordering, and auxiliary constraints
- Motivation: Provides a lifted search space, avoids re-generating the same failures multiple times (also, rich connections to combinatorial problems)
- Issues:
  - Refining disjunctive plans
    - Graphplan (Blum & Furst, 95)
  - Solution extraction in disjunctive plans
    - Direct combinatorial search
    - Compilation to CSP/SAT/ILP

We will first review some core CSP/SAT concepts and then discuss approaches to disjunctive planning

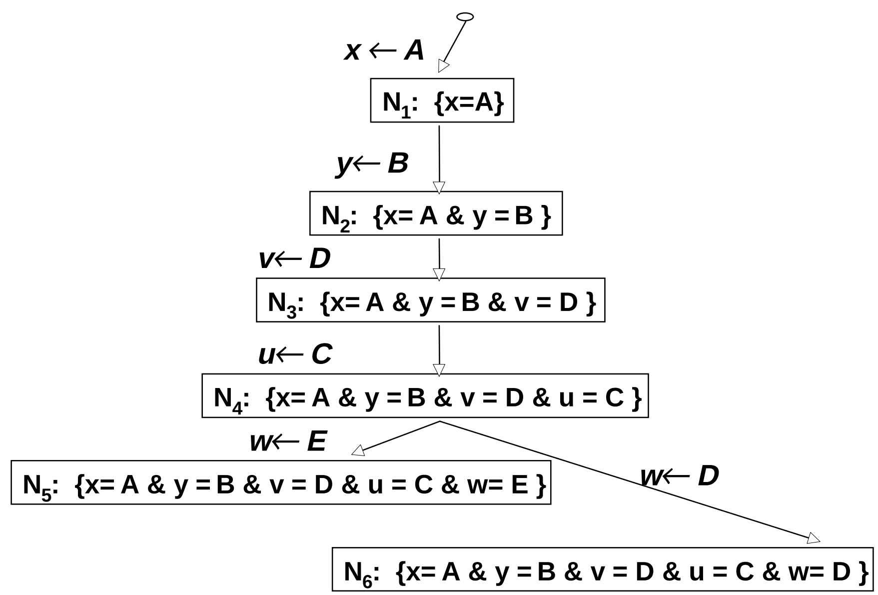
CSP and SAT Review

(very) Quick overview of CSP/SAT concepts

- **Constraint Satisfaction Problem (CSP)**
  - **Given**
    - » **A set of discrete variables**
    - » **Legal domains for each of the variables**
    - » **A set of constraints on values groups of variables can take**
  - **Find an assignment of values to all the variables so that none of the constraints are violated**
- **SAT Problem = CSP with boolean variables**

x,y,u,v: {A,B,C,D,E}
w: {D,E} l : {A,B}
$x=A \Rightarrow w \neq E$
$y=B \Rightarrow u \neq D$
$u=C \Rightarrow l \neq A$
$v=D \Rightarrow l \neq B$

A solution:
x=B, y=C, u=D, v=E, w=D, l=B

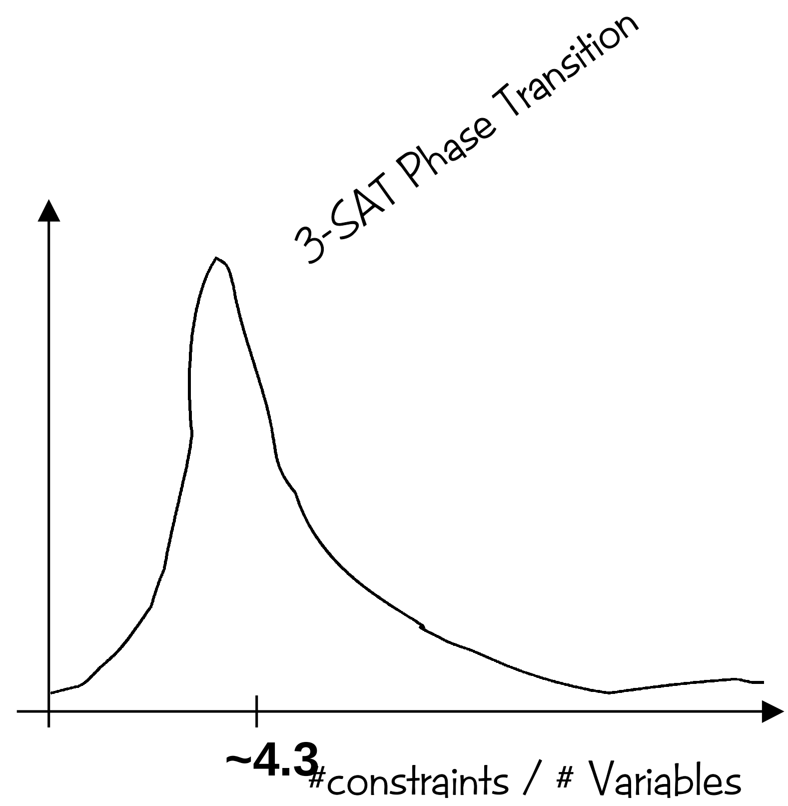


What makes CSP problems hard?

Asignments to individual variables that seem locally consistent are often globally infeasible, causing costly backtracking.

The difficulty of a CSP/SAT problem depends on

- Number of variables (propositions)
- Number of constraints (clauses)
- Degree of local consistency



Hardness & Local Consistency

- An n-variable CSP problem is said to be k-consistent iff every consistent assignment for (k-1) of the n variables can be extended to include any k-th variable
  - » Directional consistency: Assignment to first k-1 variables can be extended to the k-th variable
  - » Strongly k-consistent if it is j-consistent for all j from 1 to k
- Higher the level of (strong) consistency of problem, the lesser the amount of backtracking required to solve the problem
  - – A CSP with strong n-consistency can be solved without any backtracking
- We can improve the level of consistency of a problem by explicating implicit constraints
  - – Enforcing k-consistency is of $O(n^k)$ complexity
    - » Break-even seems to be around k=2
    - » Use of directional and partial consistency enforcement techniques

Important ideas in solving CSPs

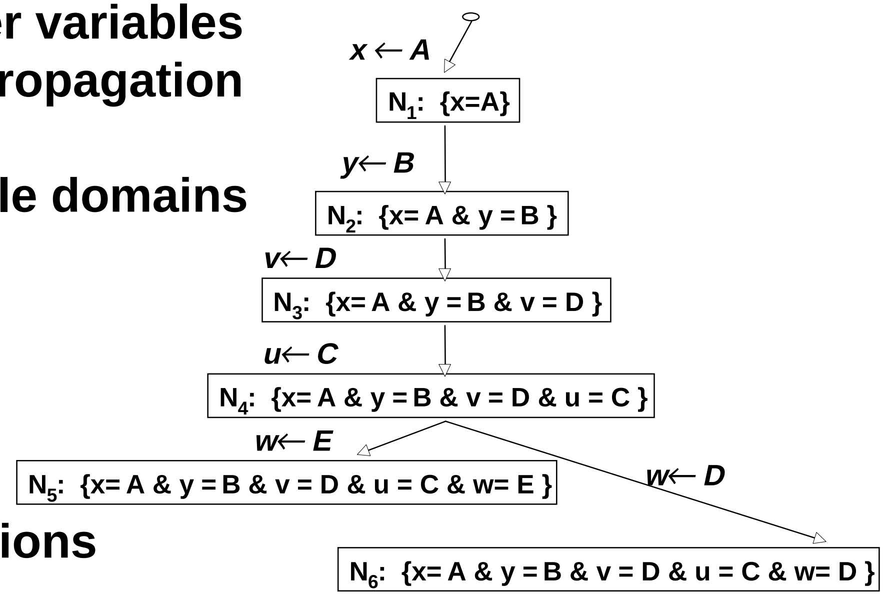
Variable order heuristics:
- Pick the variable with smallest domain
- Or the variable that constrains most other variables
- Or the variable that supports most unit propagation

Forward Checking: Filter out future variable domains based on current assignments
- --Unit propagation fills this role in SAT

DDB: Explain the failure at the dead-end nodes in terms of violated constraints, and during backtracking, skip over decisions that do not affect the explanation

EBL: Remember interior node failure explanations as additional (implied) constraints)

Local Search Methods: Change the assignment leading to most increase in the number of satisfied constraints

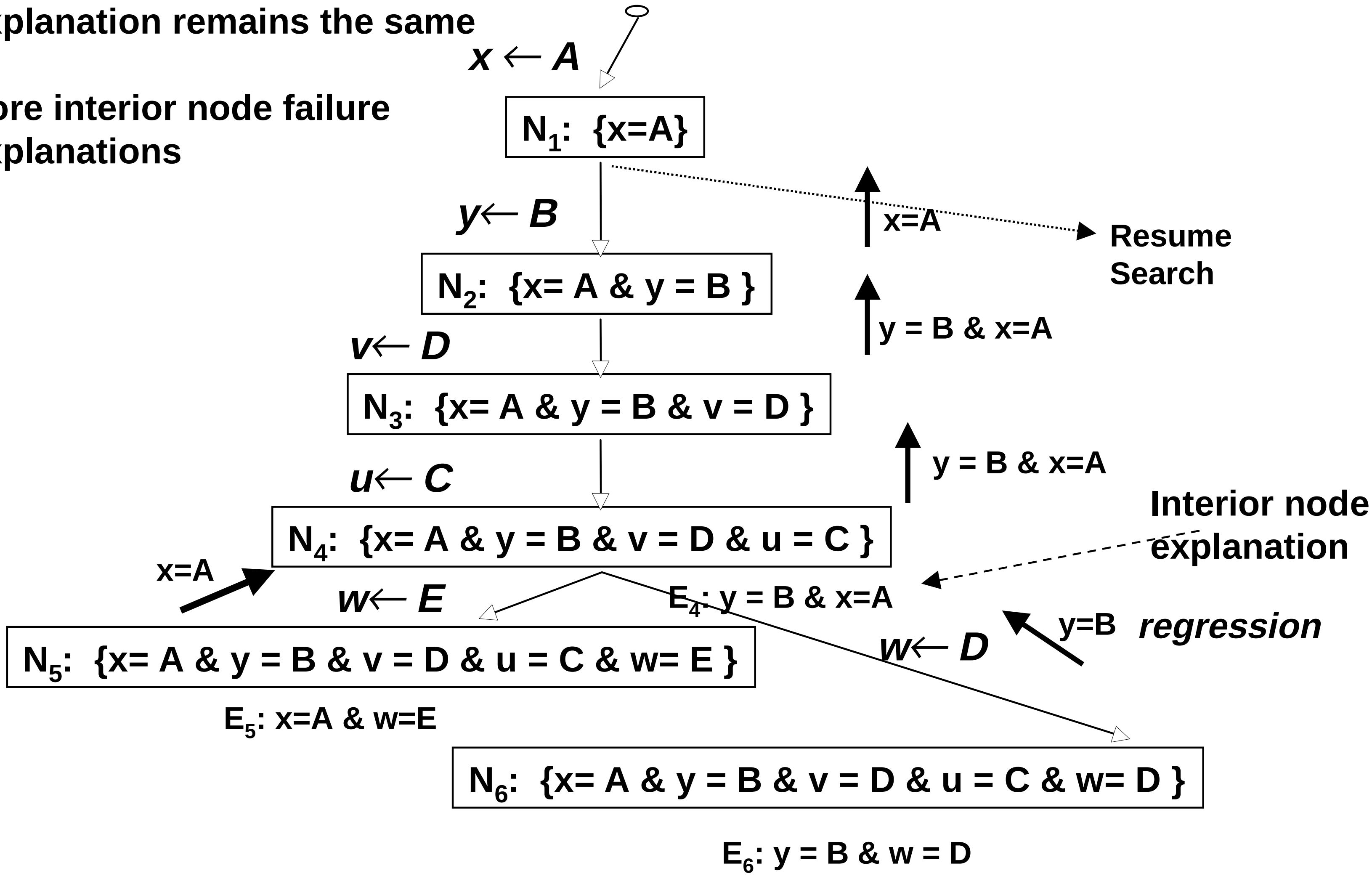


x,y,u,v: {A,B,C,D,E}
w: {D,E} l : {A,B}
$x=A \Rightarrow w \neq E$
$y=B \Rightarrow u \neq D$
$u=C \Rightarrow l \neq A$
$v=D \Rightarrow l \neq B$

DDB & EBL

DDB: Skip a level if the regressed Explanation remains the same

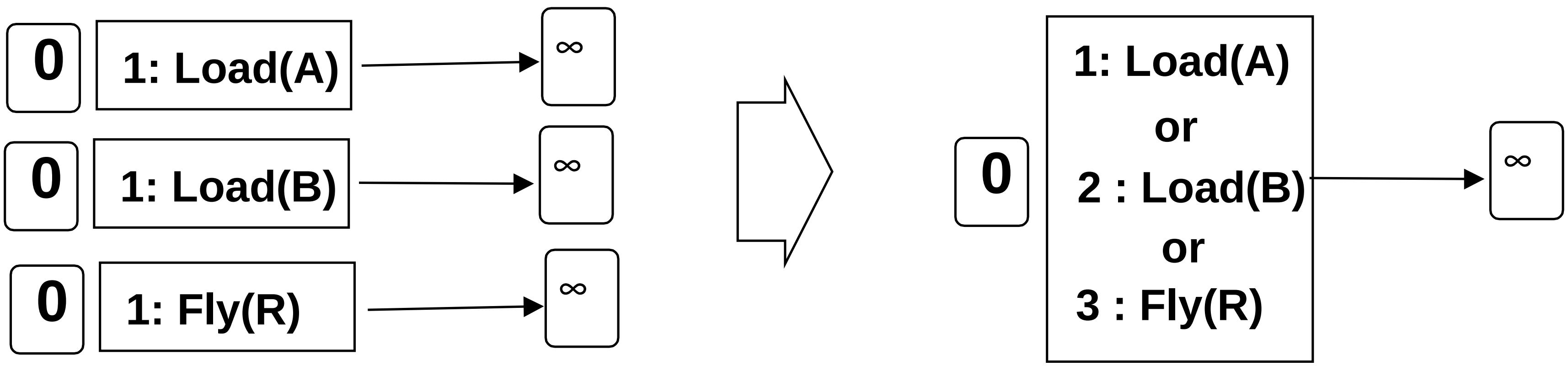
EBL: Store interior node failure Explanations

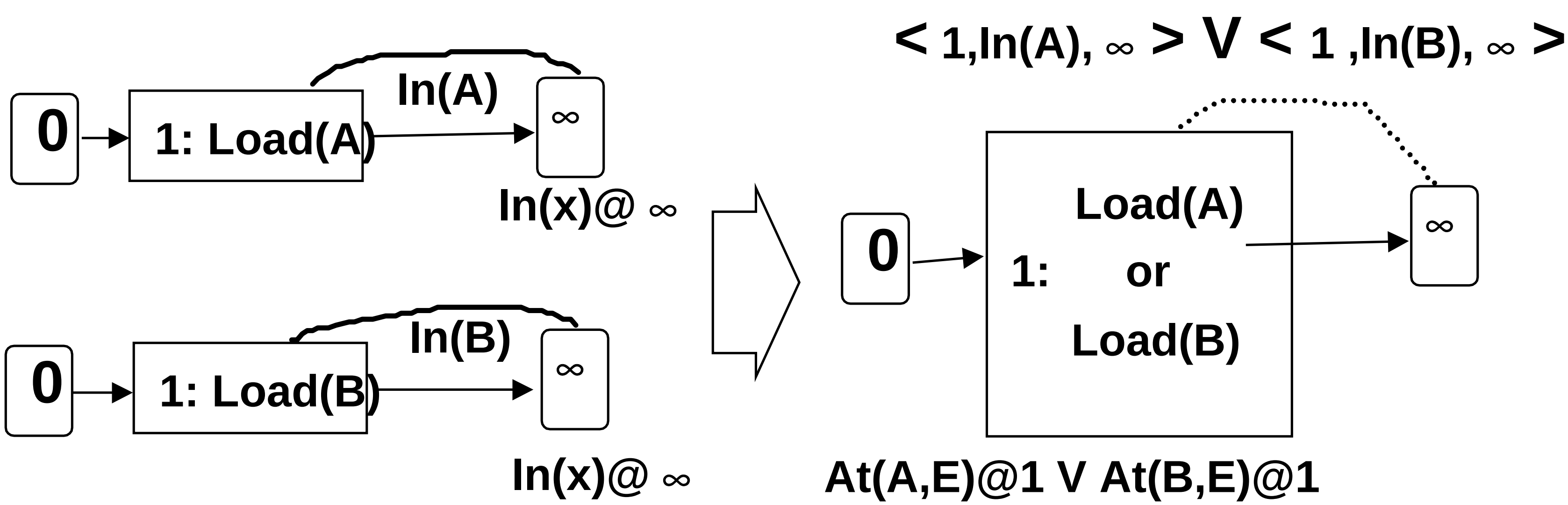


Disjunctive Planning ..contd.

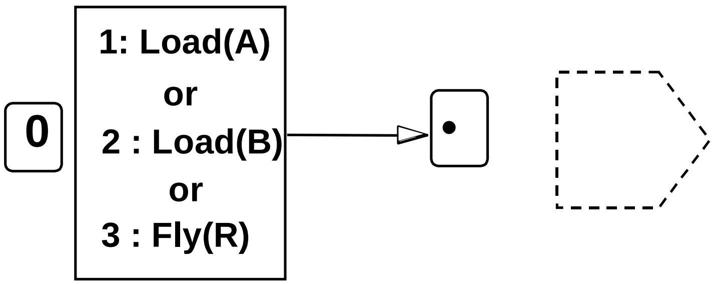
Disjunctive Representations

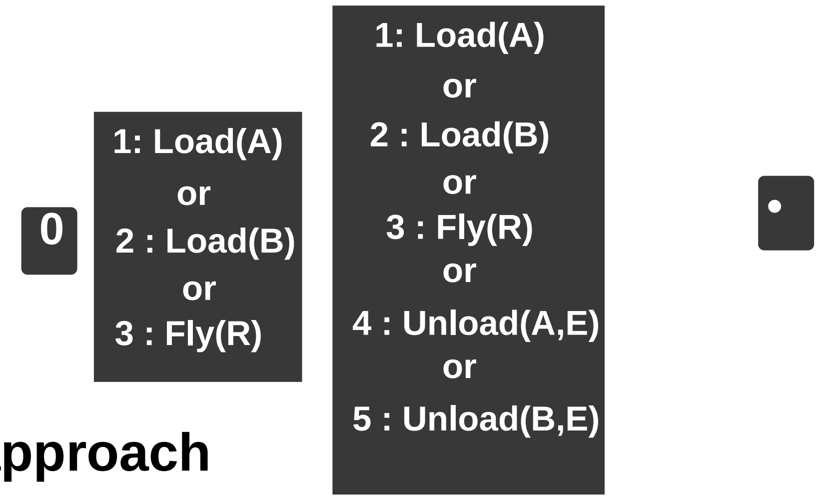
--Allow disjunctive step, ordering and auxiliary constraints in partial plans





Refining Disjunctive Plans (1)

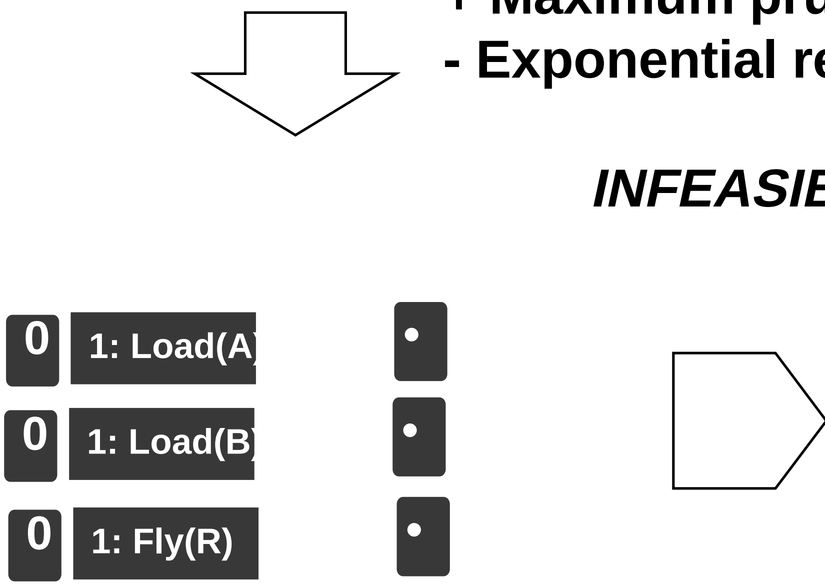


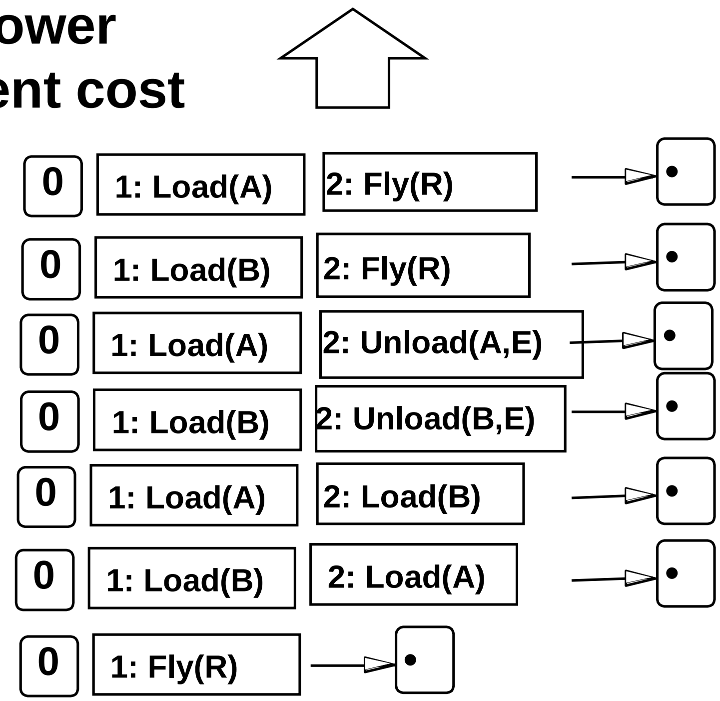


Indirect unioned approach

+ Maximum pruning power
- Exponential refinement cost

INFEASIBLE





Refining Disjunctive plans (2)

Direct naive approach

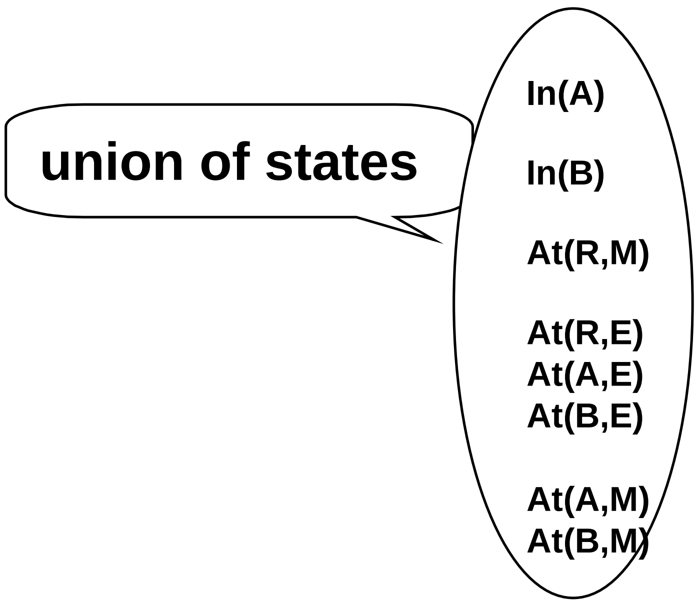
Put all actions at all levels

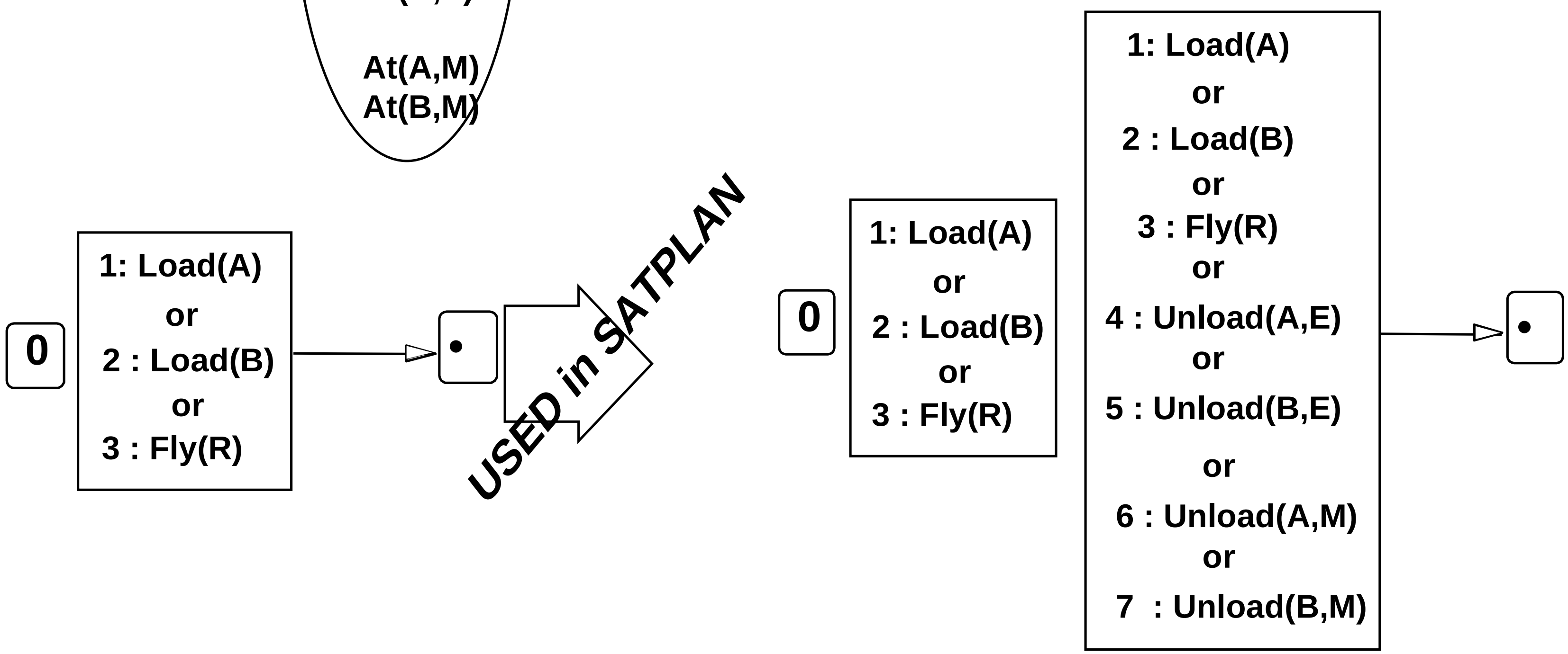
--Proposition list contains all conditions

+ Trivial to construct

- Loss of pruning power (progressivity)

=> Costlier solution extraction





Refining Disjunctive plans (3)

Enforce partial 1-consistency

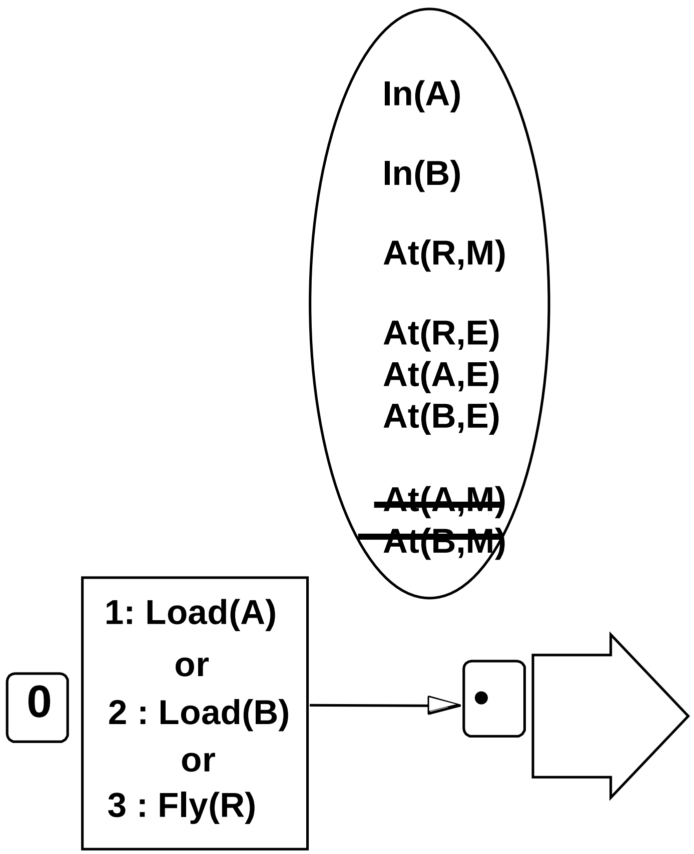
Proposition list avoids unsupported conditions

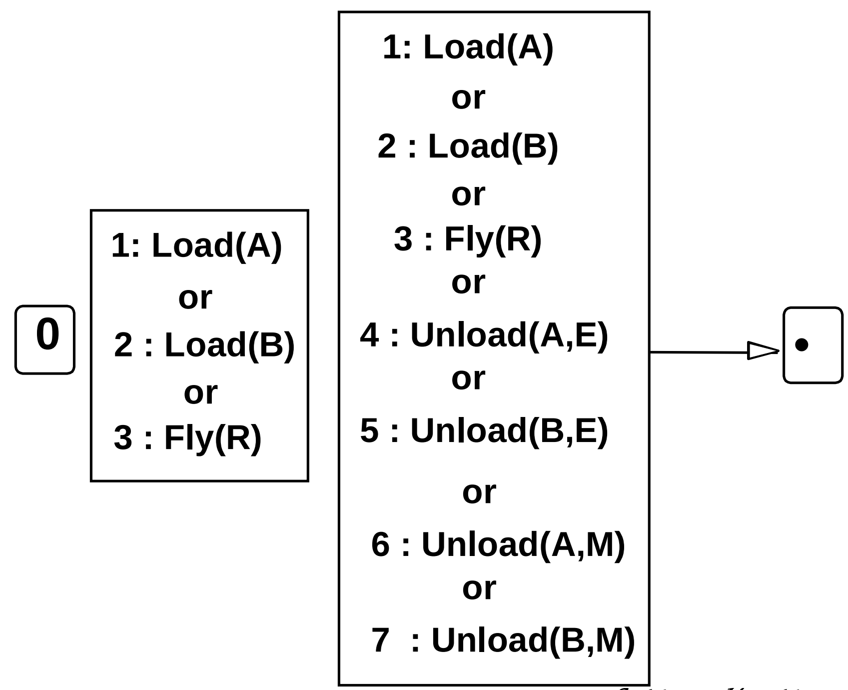
+ Polynomial refinement time

-- Loss of pruning power (progressivity)

=> too many (minimal) candidates

=> Costlier solution extraction



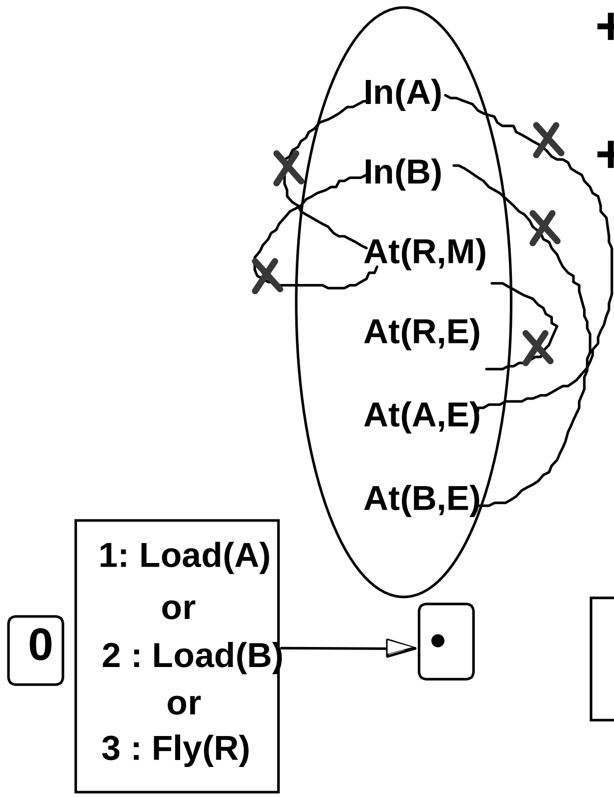


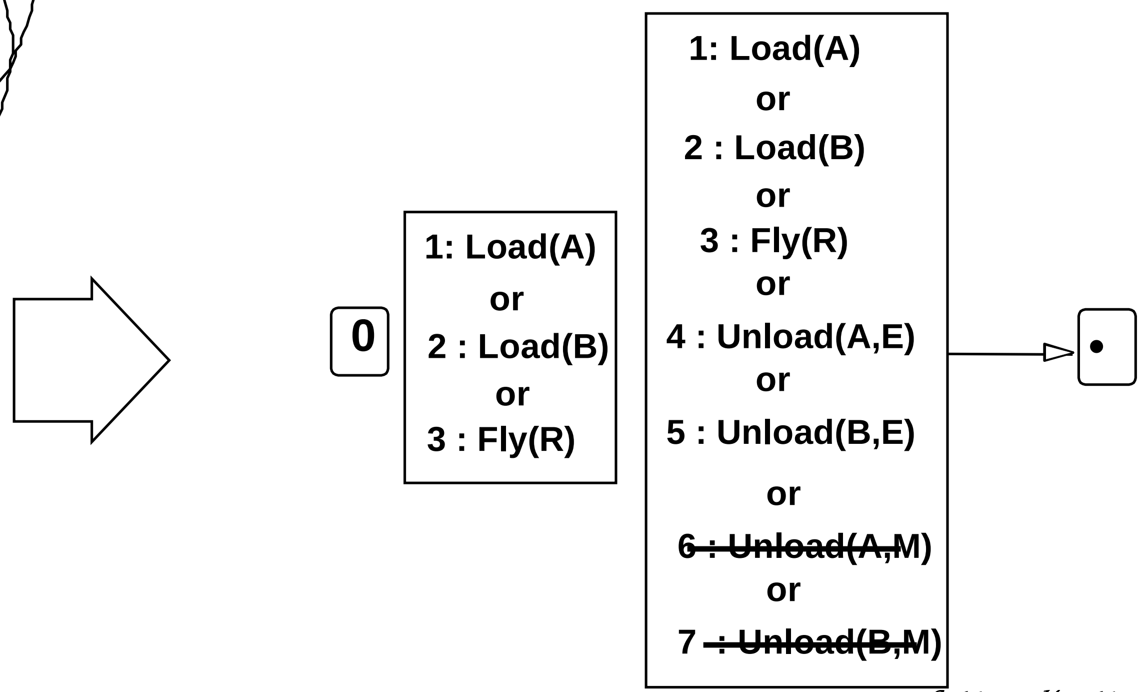
Refining Disjunctive plans (4)

Enforce (partial) 2-consistency

Proposition list maintains interactions between pairs of conditions

+ Polynomial refinement time higher pruning power
+ Better balance between refinement cost and solution extraction cost

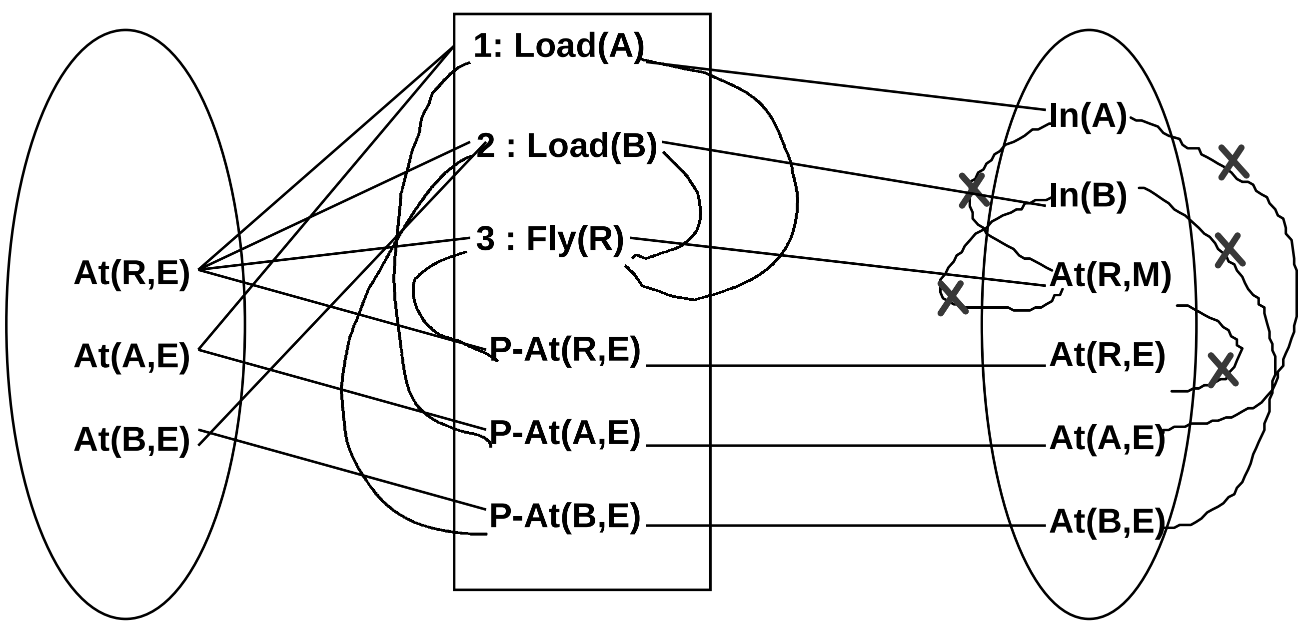




Graphplan Plangraph

(Blum & Furst, 1995)

- Graphplan directly refines disjunctive plans using forward state space refinement
  - The plan graph structure explicitly contains proposition lists, persistence actions for each condition, and dependency links between actions and propositions
  - Enforces partial 2-consistency by incrementally computing and propagating mutual exclusion relations

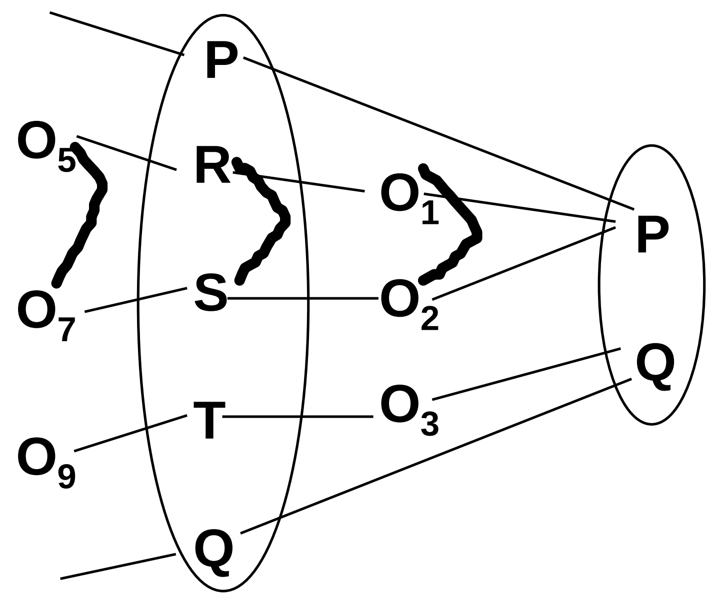


Open issues in disjunctive refinement

- **Directed partial consistency**
  - **Mutex propagation is a form of reachability analysis**
    - **Relevance analysis?**
  - **Higher levels of directional consistency?**
- **Supporting refinements other than FSR**
  - **Direct naïve refinements are easy to support; enforcing an appropriate level of consistency is harder**
  - **Some "relevance" based approaches exist for BSR**
    - **Inseperability, backward mutex (see next)**
    - **Can be used in conjunction with reachability analysis**
  - **Enforcing effective consistency for PSR is still virgin territory…**

Ideas for enforcing consistency for BSR

- **1-consistency based on relevance**
  - **Every action present in the final level will give either P or Q**
- **Pairs of conditions or actions that will not be required together**
  - **O1 or O2 is enough; so R or S is enough, so O5 or O7 is enough**
- **Sets of conditions or actions that are "*inseperable*"**
  - **P & Q are inseperable;**
  - **Set of actions supporting P is inseperable with that supporting Q**



Solution Extraction in Disjunctive Plans

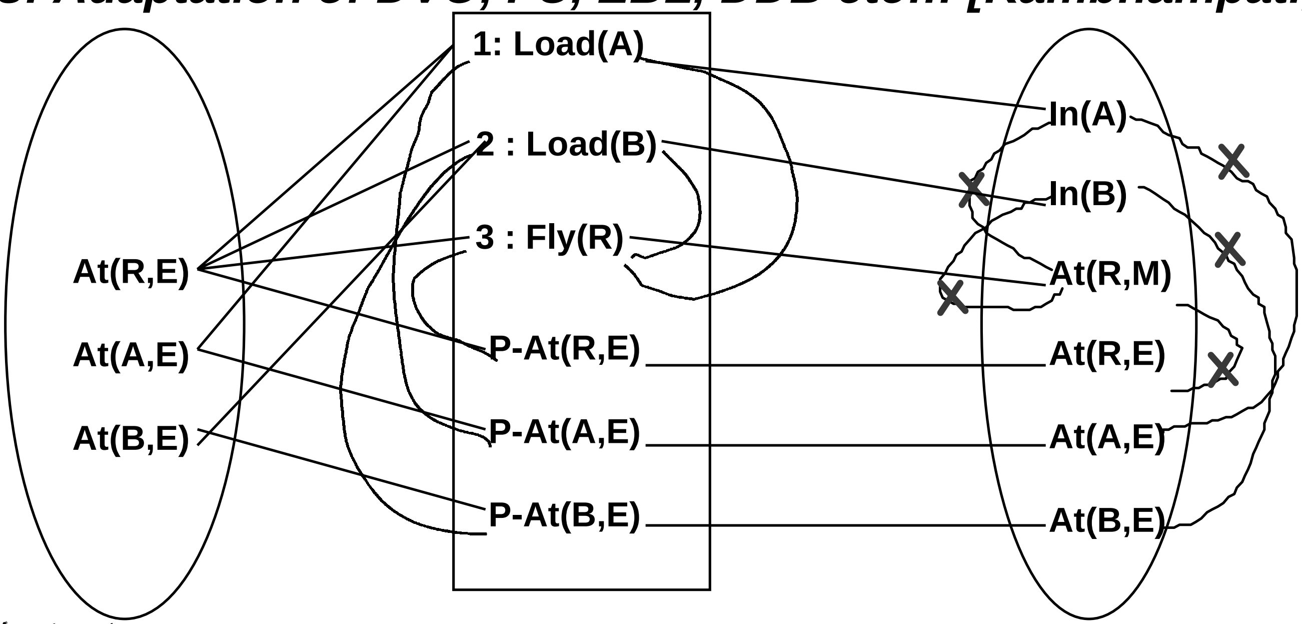
- **Even after refinement, a k-length disjunctive plan contains too many potential k-length plans.**
- **Looking for a solution plan in the disjunctive structure is a combinatorial problem**
  - **Direct methods: Can be solved using special techniques customized to the disjunctive structure**
    - **Graphplan backward search; Graphplan local search**
  - **Compilation: Can be polynomially compiled to any canonical combinatorial problems, including**
    - **Constraint Satisfaction Problem (CSP)**
    - **Boolean Satisfiability Problem (SAT)**
    - **Integer Linear Programming (ILP)**

Graphplan Backward Search
(Direct Search I)

Objective: Find a sub-graph of the plangraph that corresponds to a valid plan.

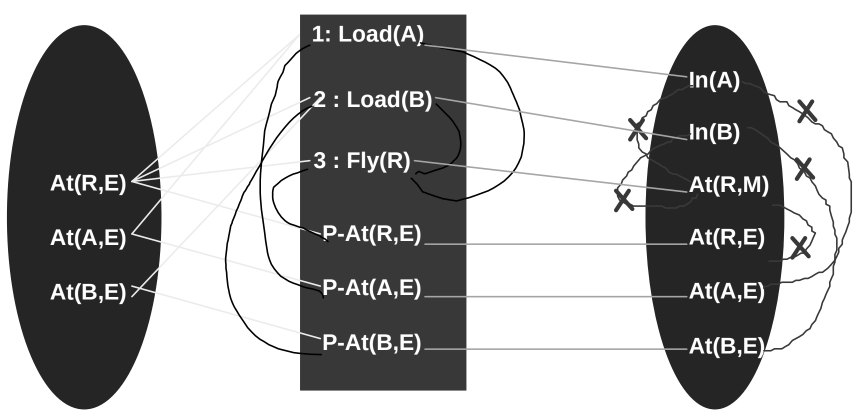
Method: Start from the goal propositions at the last level
Select actions to support the goals so that no two are *mutex* *(choice)*
Recurse on the preconditions of the selected actions
(recursion ends at the initial state)
(When backtracking over the goals at a level, memoize them)

Optimizations: Adaptation of DVO, FC, EBL, DDB etc… [Kambhampati, IJCAI99]



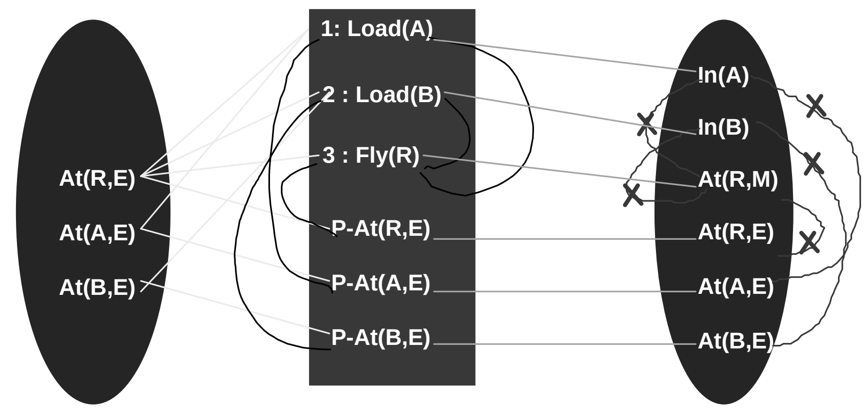
Other Direct Extraction Strategies

- **Motivation: No compelling reason for making the search for a valid subgraph backward, or systematic...**
- **Alternatives:**
  - **Forward Search (dynamic programming) [Kambhampati & Parker 98; Blum & Langford 98]**
  - **Systematic Undirectional search [Rintanen, 98]**
    - *Select an action anywhere in the plan-graph for inclusion in the solution; Propagate consequences (adapts normal CSP Search to plan-graph)*
  - **Local Search [Gerevini et. al., 99]**



Compilation to CSP

Suppose we want to find a plan that satisfies In(A) & In(B)



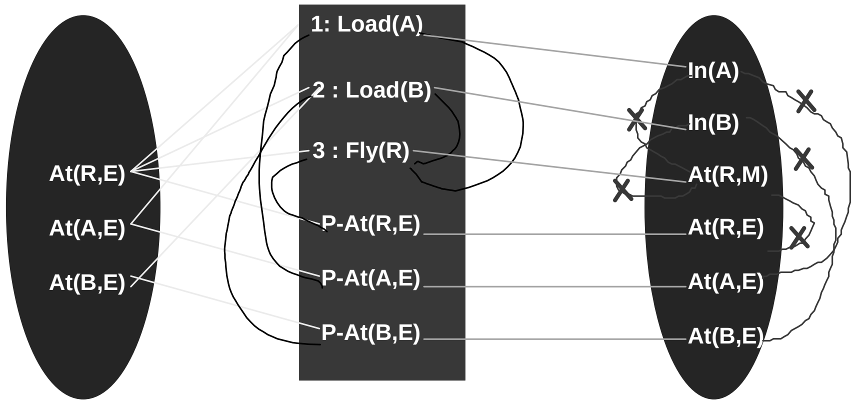
Variables: Propositions (In-A-1, In-B-1, ..At-R-E-0 …)
Domains: Actions supporting that proposition in the plan
In-A-1 : { Load-A-1, #} At-R-E-1: {P-At-R-E-1, #}

Constraints: Mutual exclusion
~[(In-A-1 = Load-A-1) & (At-R-M-1 = Fly-R-1)] ; etc..

Activation
In-A-1 != # In-B-1 != # (Goals must have action assignments)
In-A-1 = Load-A-1 => At-R-E-0 != # , At-A-E-0 != #
(subgoal activation constraints)

Compilation to SAT

Suppose we want to find a plan that satisfies In(A) & In(B)



Init: At-R-E-0 & At-A-E-0 & At-B-E-0
Goal: In-A-1 & In-B-1

Graph: *"cond at k => one of the supporting actions at k-1"*
In-A-1 => Load-A-1 In-B-1 => Load-B-1
At-R-M-1 => Fly-R-1 At-R-E-1 => P-At-R-E-1

Load-A-1 => At-R-E-0 & At-A-E-0 *"Actions => preconds"*
Load-B-1 => At-R-E-0 & At-B-E-0
P-At-R-E-1 => At-R-E-0h

~In-A-1 V ~ At-R-M-1 ~In-B-1 V ~At-R-M-1 *"Mutexes"*

Compilation to Integer Linear Programming

- **Motivations**
  - Ability to handle numeric quantities, and do optimization
  - Heuristic value of the LP relaxation of ILP problems
  - Deep connections between CSP and ILP (Chandru & Hooker, 99)
- **Conversion**
  - Explicitly set up ILP inequalities corresponding to the disjunctive plan (Bylander, 97)
  - Convert a SAT/CSP encoding to ILP inequalities
    - » E.g. X v ~Y v Z => x + (1 - y) + z >= 1
- **Solving**
  - Use LP relaxation as a heuristic (akin to Greedy Regression Graphs), or as a seed (for local search)
  - Solve using standard methods (not competitive with SAT approaches)

[Kautz, McAllester, Selman, 96]

Direct generation of SAT encodings

- **Bounded-length plan finding can be posed directly as a SAT encoding (skipping the refinement step).**
  - **Set up k-length sequence of disjunctive actions (a1 V a2 V ….V an)**
    - » **In effect, a direct naïve refinement is used (monotonic, complete, but not progressive)**
  - **Impose constraints, which when satisfied, will ensure that some sub-sequence of the disjunctive sequence is a solution to the planning problem**
    - » **The constraints set up lines of proof**
      - **State-space proofs**
      - **Causal proofs**

a1	**a1**	**a1**	**a1**
a2	**a2**	**a2**	**a2**
a3	**a3**	**a3**	**a3**
….	**….**	**….**	**….**
an	**an**	**an**	**an**
1	**2**		**k**

Encodings based on state-based proofs

Propositions corresponding to initial conditions and goals are true at their respective levels

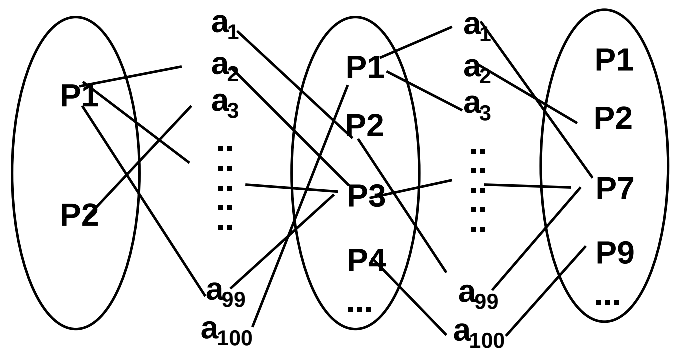
P1-0 & P2-0 & P7-k & P9-k

At least one of the actions at each level will occur

a1-j V a2-J V … V an-j

Actions imply their preconditions and effects

ai-k => $prec_{ai}$-k-1 & Eff_{ai}-k



Progression (classical Frame)

A proposition P at j remains true if no action occuring at j+1 deletes P

Pi-j & Ak-(j+1) => Pi-j

forall Ak that don't affect Pi

No more than one action occurs at each step

~aj-k V ~am-k forall j,m,k

Regression (explanatory frame)

A proposition P changes values between j and j+1 only if an action occurs that makes it so

~Pi-j & Pi-(j+1) => al-j V am-j ..

Where al, am… add Pi

No pair of interacting actions must occur together

~aj-k V ~am-k forall k

forall aj,am that interfere

Encodings based on Causal proofs

S0	S1	S2	S3	Sk	S∞
	a1	a1	a1	a1	
Needs(S0, I1)	a2	a2	a2	a2	Needs(S∞, G1)
Needs(S0, I2)	a3	a3	a3	a3	Needs(S∞, G2)
□	….	….	….	….	
Needs(S0, In)	an	an	an	an	

Each step is mapped to exactly one action
Si = A1 V Si = A2 … ; ~(Si = Ai & Si = Aj)
A step inherits the needs, adds and deletes of the action it is mapped to
Si = Aj => Adds(Si, Pa) & Needs(Si, Pp) & Deletes(Si, Pd)
A Step get needs, adds and deletes only through mapped actions
Adds(Si, Pa) => Si = Aj V Si = Ak … (Ai, Ak add Pa)
Every need is established by some step
Needs(Si,Pj) => Estab(S1, Pj, Si) V Estab(S2, Pj, Si) … V Estab(Sk, Pj, Si)

Establishment with causal links
Estab(S1,Pj,Si) => Link(S1,Pj,Si)
Link implies addition & precedence
Link(Si,Pj,Sk) => Adds(Si,Pj) & Precedes(Si,Pj)
Link implies preservation by intervening steps
Link(Si,Pj,Sk) & Deletes(Sm,Pj) => Precedes(Sm, Si) V Precedes(Sk, Si)

Precedence is irreflexive, asymmetric and transitive...

[Mali & Kambhampati, AAAI-99]

Alternative causal encodings

S0	S1	S2	S3	Sk	S∞
	a1	a1	a1	a1	
Needs(S0, I1)	a2	a2	a2	a2	Needs(S∞, G1)
Needs(S0, I2)	a3	a3	a3	a3	Needs(S∞, G2)
□	….	….	….	….	
Needs(S0, In)	an	an	an	an	

Whiteknight based establishment

Some preceding step adds
Estab(S1,Pj,Si) => Adds(S1, Pj)
& Precedes (S1, Sj)
For every preceding step deleting a needed condition, there is a white-knight that adds it back
Needs(Sj, Pj) & Deletes(Sd, Pj) & Precedes(Sd,Sj)
=> Wknight(S1, Sj, Sd, Pj) V
Wknight(S2,Sj,Sd,Pj) V ….

Wknight(Sw,Sj,Sd,Pj) => Precedes(Sw, Sj) &
Precedes(Sd, Sw) & Adds(Sw, Pj)

Contiguity based precedence

Step Sj is in position j

Precedes(S1,S2)
Precedes(S1, S3)
…….
Precedes(Sj,Sj+k)

Eliminates the need for $O(k^3)$ Transitivity clauses

Eliminates the need for $O(k^2)$ causal link variables

Simplifying SAT encodings

- Constraint propagation...
  - Unit propagation   P , ~P V Q => Q
  - Pure literal elimination (If a proposition occurs with the same polarity in any clause that it occurs in, set it to True)
    - » Works only when all satisfying assignments are considered equivalent [Wolfe & Weld, IJCAI-99]
- Syntactic manipulations:
  - Resolve away dependent variables
    - » Fluent or Action variables can be eliminated in state-based encodings
      - P11--A12--P22 becomes P11 => P22
- Compilation Tricks
  - Split n-ary actions & propositions to 2-ary ones
    - » Move(x,y,z,t) leads to O(#obj * #obj * #obj X k) variables
    - » MB(x,t), MF(y,t), M(z, t) leads to 3*O(#obj, k) variables
  - Lifting (variablized encodings)???

Tradeoffs between encodings based on different proof strategies

- Progression (classical frame) encodings lead to higher number of clauses, and allow only serial plans
  - O(#prop * #actions * #levels)
  - To allow parallel plans, we need to look at frame axioms with sets of actions, increasing the clauses exponentially
- Regression (explanatory frame) encodings reduce clauses, and allow parallel plans
  - O(#prop * #levels)
- Empirical results validate dominance of regression over progression encodings
  - The SAT compilation of Graphplan plan-graph corresponds to a form of backward encoding

Tradeoffs between encodings based on different proof strategies

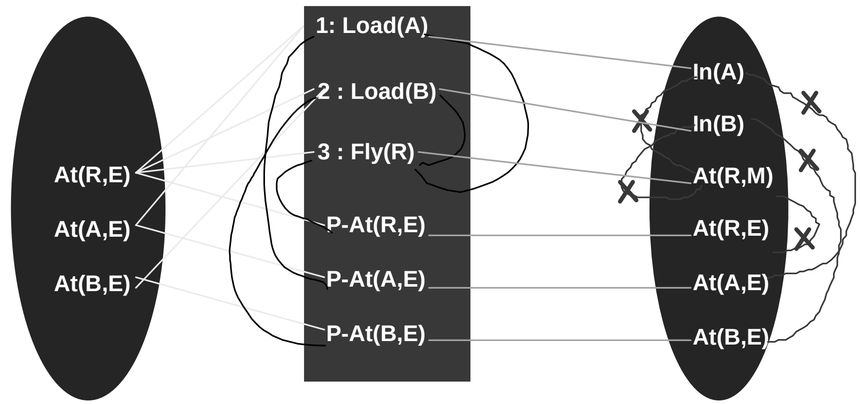
- **Causal encodings in general have more clauses than state-space encodings**
  - **O(#actions x #actions x #fluents) for causal link variables**
    - » **Could be reduced by using white-knight based proofs**
  - **O(#actions x #actions x #actions) clauses for partial ordering**
    - » **Could be reduced by using contiguous ordering**
  - **However, the best causal encodings will still be dominated by the backward state-space encodings [Mali & Kambhampati, 99]**
- **Paradoxical given the success of partial order planners in conjunctive planning?**
  - **Not really! We are using causal proof which is typically longer than state-based proofs, and are not using the flexibility of step insertion.**
    - » **Can be helpful in incremental planning & Plan reuse**
    - » **Are helpful in using causal domain specific knowledge (e.g. HTN schemas)**

Direct compilation vs. compilation of refined disjunctive plans

- **Direct encodings correspond to translation of Direct Naïve refinements**
  - **Non-progressive, have large number of minimal candidates compared to encodings based on refined disjunctive plans (such as Graphplan plan-graph)**
  - **Enforcing consistency at SAT level is costly (lack of direction)**
- **And yet, SATPLAN, that used direct SAT encodings did better than Graphplan, that worked with plangraph…**
  - **Paradox? NOT.. It just shows that the solution extraction through SAT was better than through vanilla Graphplan backward search! [Kambhampati, IJCAI-97; Rintanen, KR-98]**
    - » **Blackbox [IJCAI-99] uses encodings based on planning graph**

On the difficulty of enforcing directional consistency at the SAT level

- Partial consistency that is enforced by refinement of disjunctive plans can also be done at the SAT level through resolution
  - But the resolution will be undirected
    - » will consider clauses from multiple levels
  - and thus can be costly
- Moral: Refinements allow *directed* consistency enforcement..

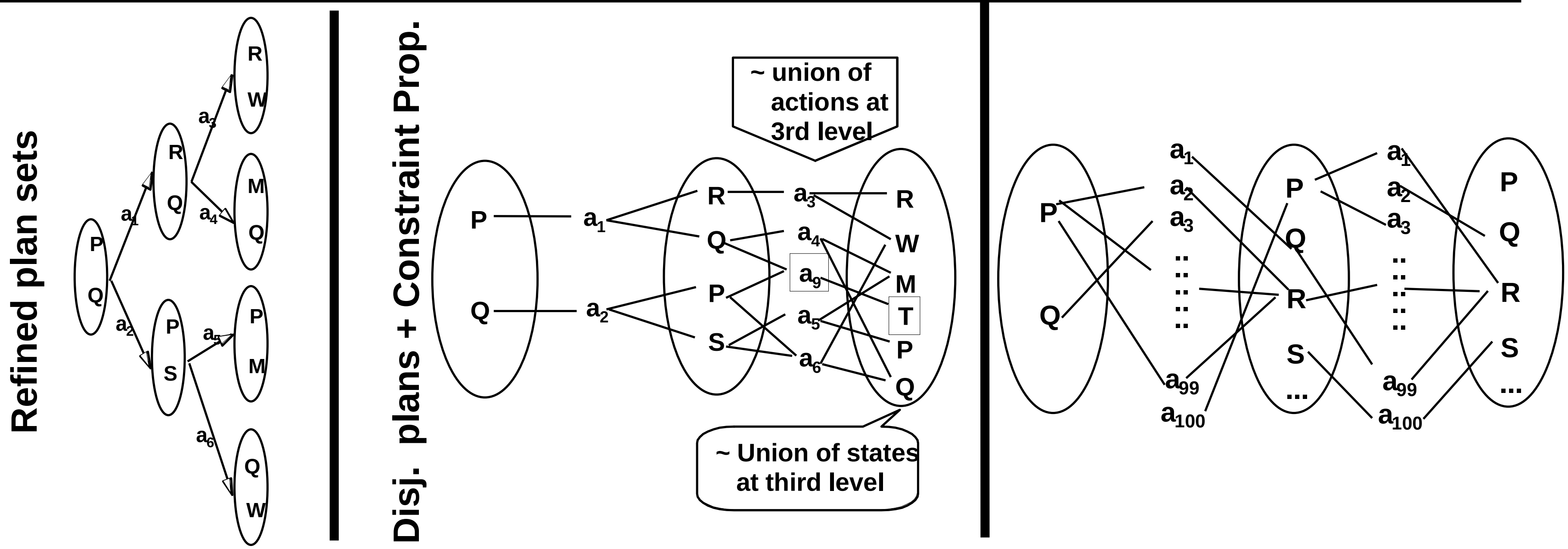


~ load-A-1 V ~load-B-1]
in-A-1 => load-A-1
in-B-1 => load-B-1

With resolution, we get
~In-A-1 V ~In-A-2

Impact of Refinements on Encoding Size

Encodings based on "refined" plans | **Direct SAT Encoding**



Encoding size increases & Cost of generating encoding reduces

-- Encodings based on refined plans can be more compact

-- *Smaller clauses, fewer variables ...*

Direct vs. compiled solution extraction

DIRECT

- ✗ Need to adapt CSP/SAT techniques
- ✓ Can exploit approaches for compacting the plan
- ✓ Can make the search incremental across iterations

Compiled

- ✓ Can exploit the latest advances in SAT/CSP solvers
- ✗ Compilation stage can be time consuming, leads to memory blow-up
- ✗ Makes it harder to exploit search from previous iteractions
- ✓ Makes it easier to add declarative control knowledge

Some implemented disjunctive planners

	Refinement	Solution Extraction
Graphplan (Blum & Furst) -- IPP (Koehler et. al.) -- STAN (Fox et. al.) --GP-EBL (Kambhampati)	***Partially 2-consistent direct refinement*** **using FSR**	**Direct search on the disjunctive plan (+ adaptation of CSP techniques such as DDB, EBL, DVO, FC etc)**
SATPLAN (Kautz & Selman)	***Naïve direct refinement*** **with FSR, BSR, PSR**	**Compilation to SAT**
Blackbox (Kautz & Selman)	***Same as Graphplan***	**Compilation to SAT**

Accelerating Disjunctive Planners

- **Reduce the size of the disjunctive plan**
  - **Relevance based pruning**
    - » **Backward plan growth [Kambhampati et. al., 97]**
    - » **Preprocessing to remove irrelevant actions and conditions [RIFO, Nebel et. al. 97]**
- **Increase the consistency level of the disjunctive plan**
  - **Learn (or input) higher-order mutexes (invariants) [Gerevini, 98]**
- **Improve the solution extraction process**
  - **Exploit Symmetry [Fox et. al. 99, Srivastava et. al. 99]**

Conjunctive vs. Disjunctive planners

- Progress depends on the effectiveness of the heuristics for judging the goodness of a partial plan
- Search control may be easier to articulate
- Space consumption can be regulated
- Better fit with mixed-initiative, incremental planning scenarios(?)

- Space consumption is a big issue
  - Creation and storage of disjunctive structures
    - The AT&T benchmark experiments were done on a 8-gig RAM m/c!
    - E.g. TLPlan beats disjunctive planners at 800-block problems…
- Better integration with non-propositional reasoners (?)

Hybrid planners--Controlled splitting &
Search in the space of disjunctive plans

Controlled Splitting

Refine (*P* : Plan)

0*. **If *P* is inconsistent, prune it. End.**
1. **If a minimal candidate corresponds to a solution, terminate.**

2. **Apply a refinement strategy R to *P* to get a new plan set *P'***

3. **Reduce components of *P'* by introducing disjunction**
4. **Propagate constraints in the components**

5. **Non-deterministically select a component P'_i of *P'***
   Call Refine(*P'*)

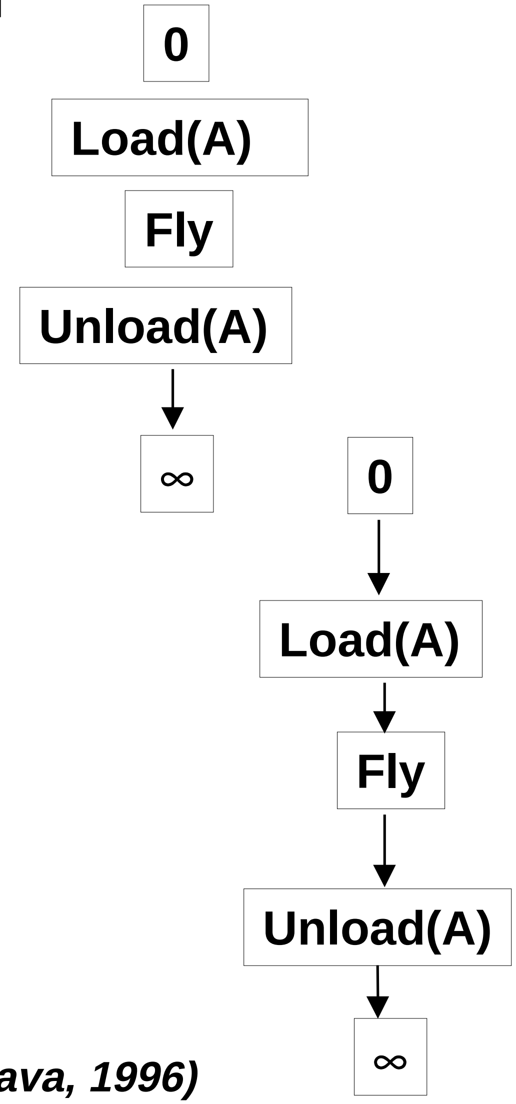
DESCARTES [Joslin & Pollack; 1995]
UCPOP-D [Kambhampati & Yang; KR, 1996]
Just scratch the surface...

Characterizing Difficult Problem Classes

- **There is a nice theory of Serializability for conjunctive planning**
- **There is no clean theory yet of what makes a problem easy or hard for disjunctive planners**
  - **Hardness characterizations in CSP/SAT can be used.**

Subgoal Interactions and Planner Selection

- ✧ **Every refinement planner R can be associated with a class $\mathcal{C}_R$ of partial plans**
- ✧ **G_1 and G_2 are trivially serializable w.r.t. a plan class $\mathcal{C}$ if every subplan $P_{Gi} \in \mathcal{C}$ is extendable to a plan for solving both.**
  - **commitment$\downarrow \Rightarrow$ trivial serializability $\uparrow$**
  - **commitment $\downarrow \Rightarrow$ plan handling cost$\uparrow$**
- ⇨ **Select the planner producing the class of <u>highest</u> commitment plans w.r.t. which most problems are trivially serializable**



(Korf, 1987; Barrett & Weld, 1994; Kambhampati, Ihrig and Srivastava, 1996)

CUSTOMIZING PLANNERS WITH DOMAIN SPECIFIC KNOWLEDGE

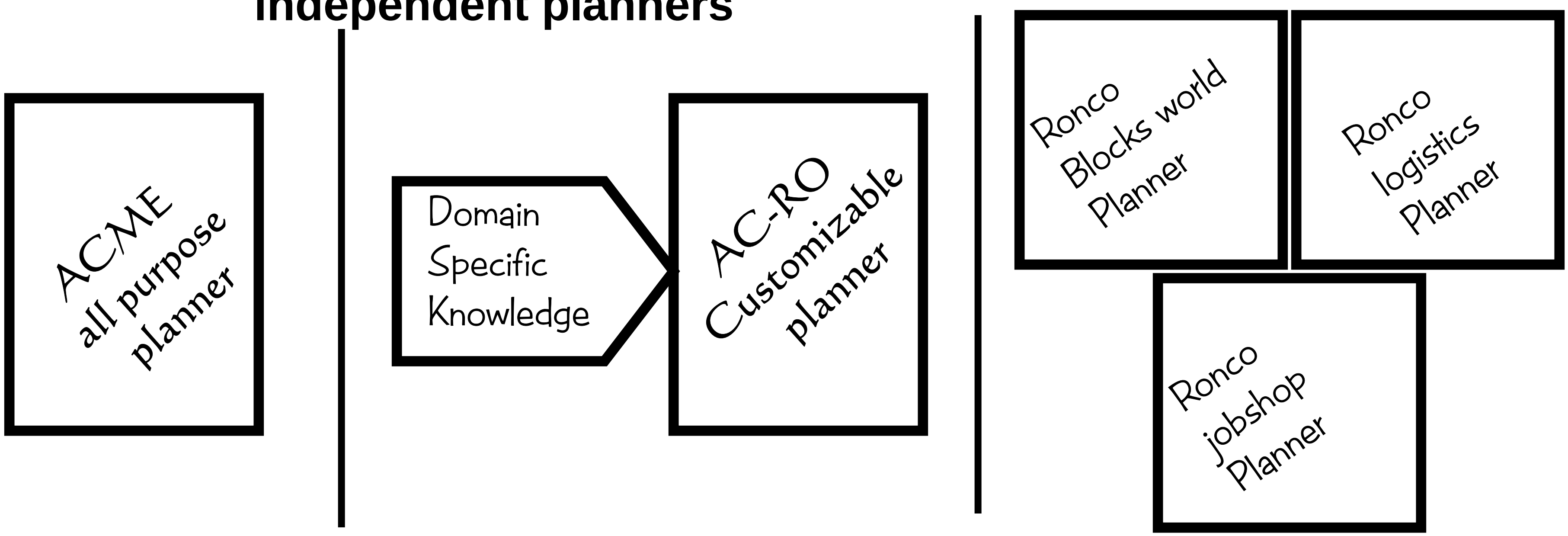
1. **User-assisted customization**
   (accept domain-specific knowledge as input)

2. **Automated customization**
   (learn regularities of the domain through experience)

User-Assisted Customization (using domain-specific Knowledge)

- Domain independent planners tend to miss the regularities in the domain
- Domain specific planners have to be built from scratch for every domain

An *"Any-Expertise"* Solution: Try adding domain specific control knowledge to the domain-independent planners



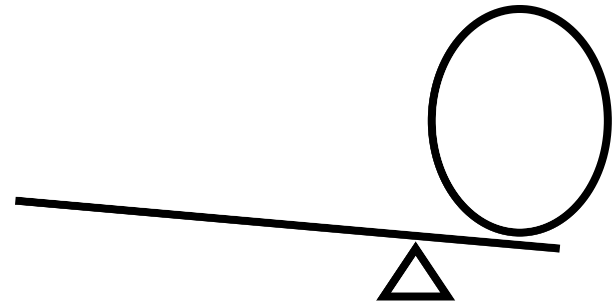
Many User-Customizable Planners

- Conjunctive planners
  - HTN planners
    - » SIPE [Wilkins, 85-]
    - » NONLIN/O-Plan [Tate et. al., 77-]
    - » NOAH [Sacerdoti, 75]
    - » Also SHOP (Nau et. al., IJCAI-99)
  - State-space planners
    - » TLPlan [Bacchus & Kabanza, 95]
  - Customization frameworks
    - » CLAY [Srivastava & Kambhampati, 97]
  - Planning & Learning
    - » Prodigy+EBL , UCPOP+EBL, DerSNLP+EBL ..
- Disjunctive planners
  - HTN SAT [Mali & Kambhampati, 98]
  - SATPLAN+Dom [Kautz & Selman, 98]

How do they relate?
What are the tradeoffs?

With enough domain knowledge any level of performance can be achieved...

- **HTN-SAT, SATPLAN+DOM beat SATPLAN…**
  - **Expect reduction schemas, declarative knowledge about inoptimal plans**
- **TLPLAN beats SATPLAN, GRAPHPLAN**
  - **But uses quite detailed domain knowledge**
- **SHOP beats TLPLAN…**
  - **Expects user to write a “program” for the domain in its language**
    - **Explicit instructions on the order in which schemas are considered and concatenated**



Who gets the credit?
-The provider of domain knowledge?
-The planner that is capable of using it?

Types of domain-specific knowledge

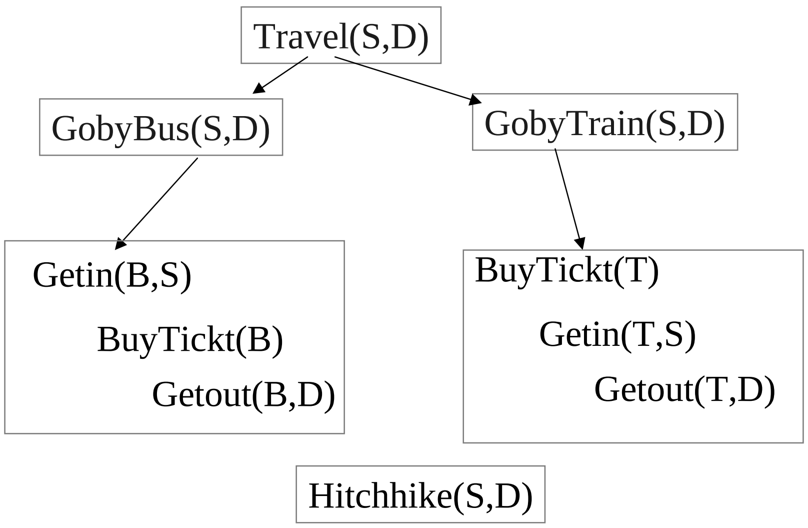
- **Declarative knowledge about desirable or undesirable solutions and partial solutions (SATPLAN+DOM)**
- **Declarative knowledge about desirable/undesirable search paths (TLPlan)**
- **A declarative grammar of desirable solutions (HTN)**
- **Procedural knowledge about how the search for the solution should be organized (SHOP)**

Uses of Domain-specific knowledge

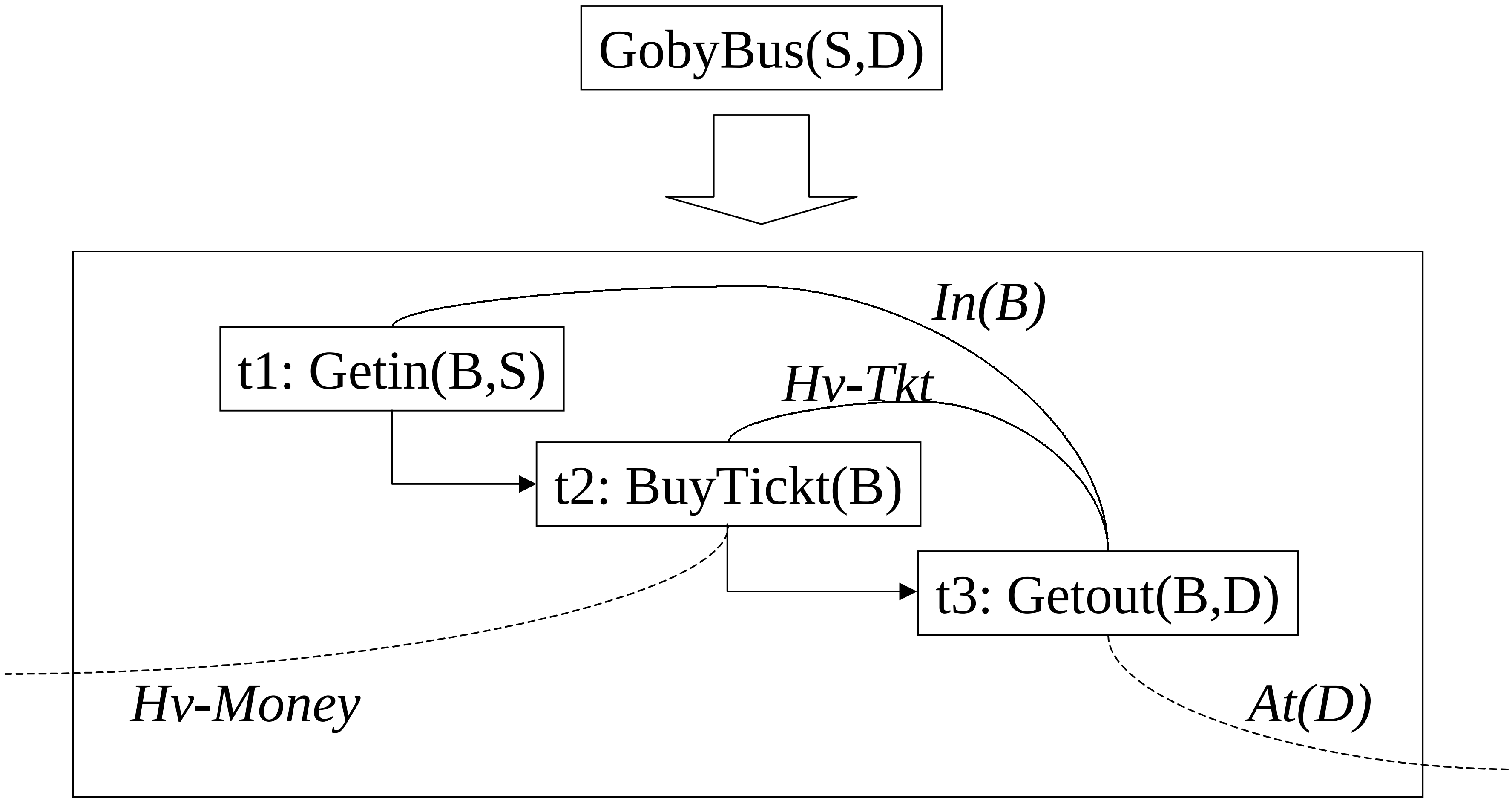
- As declarative search control
  - HTN schemas, TLPlan rules
- As procedural guidance
  - (SHOP)
- As declartative axioms that are used along with other knowledge
  - SATPlan+Domain specific knowledge
- Folded into the domain-independent algorithm to generate a new domain-customized planner
  - CLAY

Task Decomposition (HTN) Planning

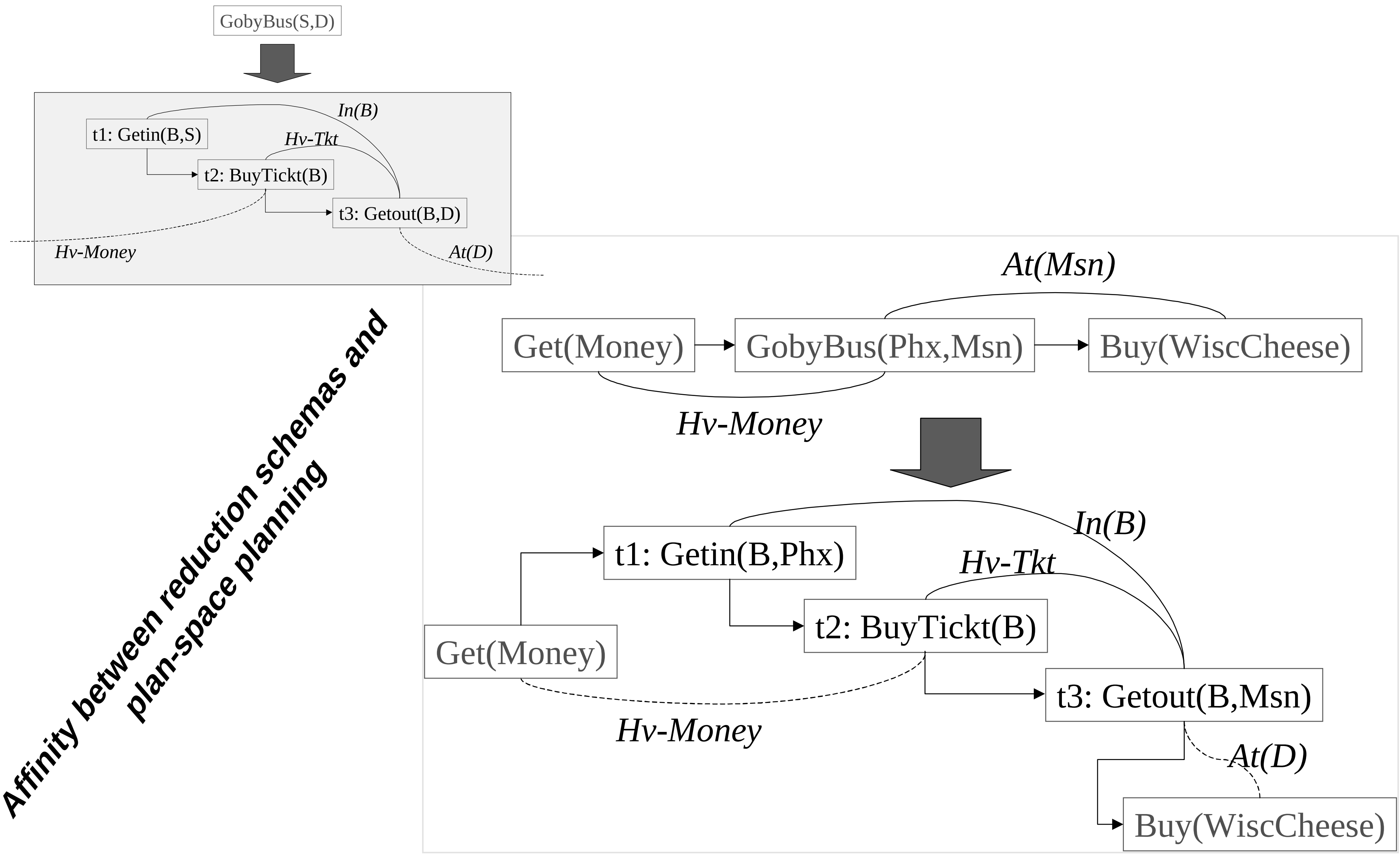
- The OLDEST approach for providing domain-specific knowledge
  - Most of the fielded applications use HTN planning
- Domain model contains non-primitive actions, and schemas for reducing them
- Reduction schemas are given by the designer
  - Can be seen as encoding user-intent
    - *Popularity of HTN approaches a testament of ease with which these schemas are available?*
- Two notions of completeness:
  - Schema completeness
    - (Partial Hierarchicalization)
  - Planner completeness



Modeling Reduction Schemas



Modeling Action Reduction



Affinity between reduction schemas and plan-space planning

Dual views of HTN planning

- Capturing hierarchical structure of the domain
  - Motivates top-down planning
    - » Start with abstract plans, and reduce them
- Many technical headaches
  - Respecting user-intent, maintaining systematicity and minimality

    [Kambhampati et. al. AAAI-98]
    - » Phantomization, filters, promiscuity, downward-unlinearizability..

- Capturing expert advice about desirable solutions
  - Motivates bottom-up planning
    - » Ensure that each partial plan being considered is "legal" with respect to the reduction schemas
    - » Directly usable with disjunctive planning approaches
- Connection to efficiency is not obvious

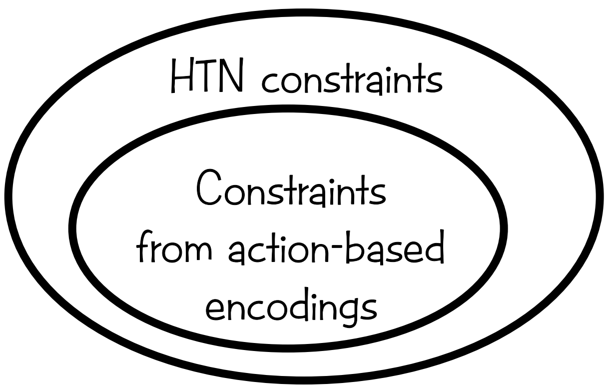
Relative advantages are still unclear...

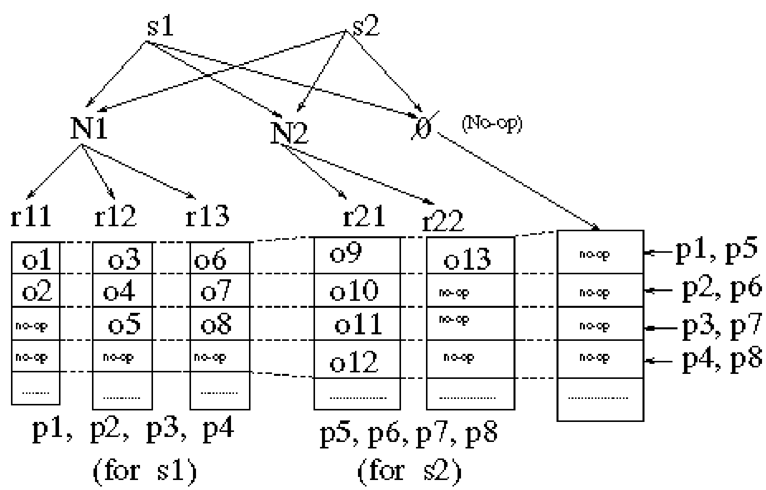
[Barrett, 97]

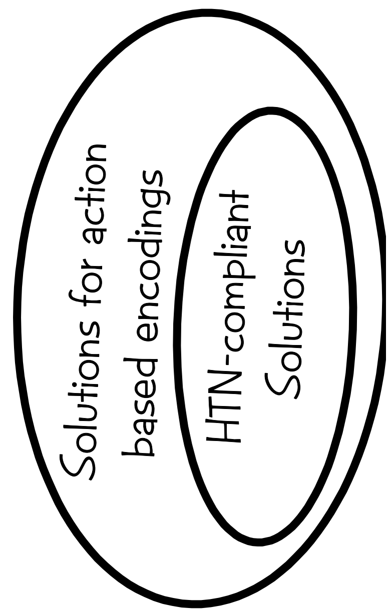
[Mali & Kambampati, AIPS-98]

SAT encodings of HTN planning

- **Abstract actions can be seen as disjunctive constraints**
  - **K-step encoding has each of the steps mapped to a disjunction of the non-primitive tasks**
  - **If a step s is mapped to a task N, then one of the reductions of N must hold (**The heart of encoding setup**)**
  - **+ The normal constraints of primitive action-based encoding**
    - » **Causal encodings seem to be a natural fit (given the causal dependencies encoded in reduction schemas)**







Solving HTN Encodings

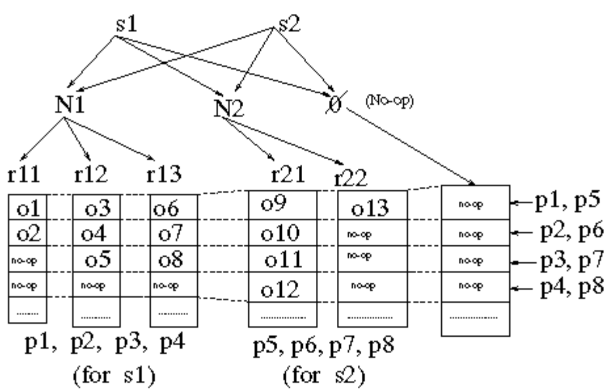
Puzzle: How can increasing encoding sizes lead to efficient planning?

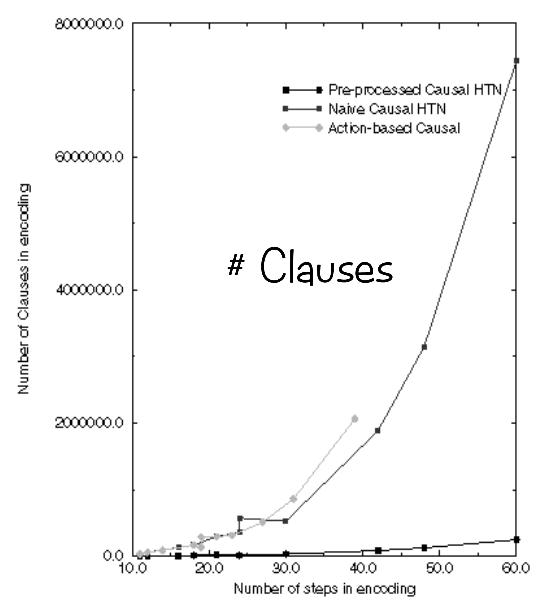
Abstract actions and their reductions put restrictions on the amount of step-action disjunction at the primitive level.

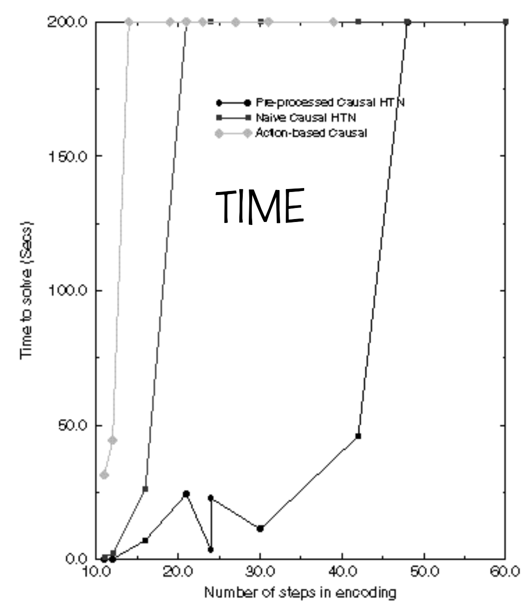
--Reduction in step-action disjunction propagates

e.g. Fewer causal-link variables, Fewer exclusion clauses…

Savings won't hold if each non-primitive task has MANY reductions







[Kautz & Selman, AIPS-98]

Non-HTN domain knowledge for SAT encodings

Invariants: *A truck is at only one location*

at(truck, loc1, I) & loc1 != loc2 => ~at(truck, loc2, I)

Optimality: *Do not return a package to a location*

at(pkg, loc, I) & ~at(pkg,loc,I+1) & I<J => ~at(pkg,loc,j)

Simplifying: *Once a truck is loaded, it should immediately move*

~in(pkg,truck,I) & in(pkg,truck,I+1) & at(truck, loc, I+1) =>
~at(truck, loc, I+2)

Once again, additional clauses first increase the encoding size but make them easier to solve after simplification (unit-propagation etc).

Rules on desirable State Sequences: TLPlan approach

TLPlan [Bacchus & Kabanza, 95/98] controls a forward state-space planner

Rules are written on state sequences using the linear temporal logic (LTL)

LTL is an extension of prop logic with temporal modalities

U	**until**	**[]**	**always**
O	**next**	**<>**	**eventually**

Example:

If you achieve on(B,A), then preserve it until On(C,B) is achieved:

[] (on(B,A) => on(B,A) U on(C,B))

TLPLAN Rules can get quite boroque

Good towers are those that do not violate any goal conditions

$$goodtower(x) \stackrel{\triangle}{=} clear(x) \wedge goodtowerbelow(x)$$

$$\begin{aligned} goodtowerbelow(x) \stackrel{\triangle}{=} & (ontable(x) \wedge \neg \mathrm{GOAL}(\exists[y{:}on(x,y)] \vee holding(x))) \\ & \vee \exists[y{:}on(x,y)]\, \neg \mathrm{GOAL}(ontable(x) \vee holding(x)) \wedge \neg \mathrm{GOAL}(clear(y)) \\ & \quad \wedge \forall[z{:}\mathrm{GOAL}(on(x,z))]\, z = y \wedge \forall[z{:}\mathrm{GOAL}(on(z,y))]\, z = x \\ & \quad \wedge goodtowerbelow(y) \end{aligned}$$

Keep growing "good" towers, and avoid "bad" towers

$$\begin{aligned} \Box \Big(& \forall[x{:}clear(x)]\, goodtower(x) \Rightarrow \bigcirc goodtowerabove(x) \\ & \wedge badtower(x) \Rightarrow \bigcirc(\neg\exists[y{:}on(y,x)]\,) \\ & \wedge (ontable(x) \wedge \exists[y{:}\mathrm{GOAL}(on(x,y))] \neg goodtower(y)) \\ & \qquad \Rightarrow \bigcirc(\neg holding(x)) \Big) \end{aligned}$$

How "Obvious" are these rules?

The heart of TLPlan is the ability to *incrementally* and *effectively* evaluate the truth of LTL formulas.

[Nau et. al., 99]

Full procedural control: The SHOP way

Shop provides a "high-level" programming language in which the user can code his/her domain specific planner

-- Similarities to HTN planning
-- Not declarative (?)

The SHOP engine can be seen as an interpreter for this language

```
(:method (travel-to ?y)
   (:first (at ?x)
    (at-taxi-stand ?t ?x)
    (distance ?x ?y ?d)
    (have-taxi-fare ?d))
  `((!hail ?t ?x) (!ride ?t ?x ?y)
    (pay-driver ,(+ 1.50 ?d)))
   ((at ?x) (bus-route ?bus ?x ?y))
  `((!wait-for ?bus ?x)
    (pay-driver 1.00)
    (!ride ?bus ?x ?y)))
```

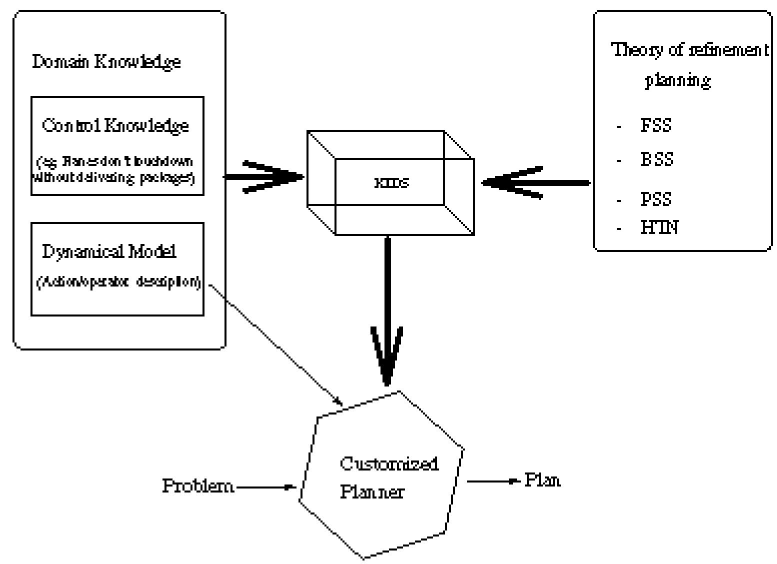
Travel by bus only if going by taxi doesn't work out

Blurs the domain-specific/domain-independent divide

How often does one have this level of knowledge about a domain?

Folding the Control Knowledge into the planner: CLAY approach

- Control knowledge similar to TLPlan's
- Knowledge is folded using KIDS semi-automated software synthesis tool into a generic refinement planning template
  - Use of program optimizations such as
    - » Finite differencing
    - » Context-dependent & independent simplification
- Empirical results demonstrate that folding can be better than interpreting rules



Caveat: Current automated software synthesis tools have a very steep learning curve

Conundrums of user-assisted cutomization

- Which planners are easier to control?
  - Conjunctive planners are better if you have search control knowledge
    - » Forward State Space (according to TLPlan)
    - » Plan-space planners (according to HTN approaches)
  - Disjunctive planners are better if your knowledge can be posed as additional constraints on the valid plans
    - » Which SAT encoding?
      - HTN knowledge is easier to add on top of causal encodings
- Which approach provides the best language for expressing domain knowledge for the lay user?
  - *(Mine--no, Mine!*)
- What type of domain knowledge is easier to validate?
- When does it become “cheating”/ “wishful-thinking”
  - Foolish not to be able to use available knowledge
  - Wishful to expect deep procedural knowledge...

Automated Customization of Planners

- **Domain pre-processing**
  - **Invariant detection; Relevance detection; Choice elimination, Type analysis**
    - » **STAN/TIM, DISCOPLAN etc.**
    - » **RIFO; ONLP**
- **Abstraction**
    - » **ALPINE; ABSTRIPS etc.**
- **Learning Search Control rules**
    - » **UCPOP+EBL,**
    - » **PRODIGY+EBL, (Graphplan+EBL)**
- **Case-based planning (plan reuse)**
    - » **DerSNLP, Prodigy/Analogy**

Most of the work has been done in the context of Conjunctive Refinement Planners

Symmetry & Invariant Detection

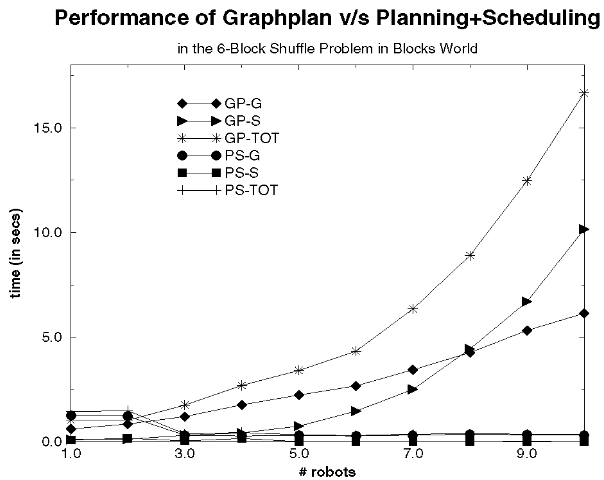
- Generate potential invariants and test them
  - DISCOPLAN [Gerevini et. al.]
    - » Allows detection of higher-order mutexes
  - Rintanen's planner
    - » Uses model-verification techniques
  - STAN/TIM
    - » Type analysis of the domain is used to generate invariants
  - ONLP (Peot & Smith)
    - » Use operator graph analysis to eliminate non-viable choices

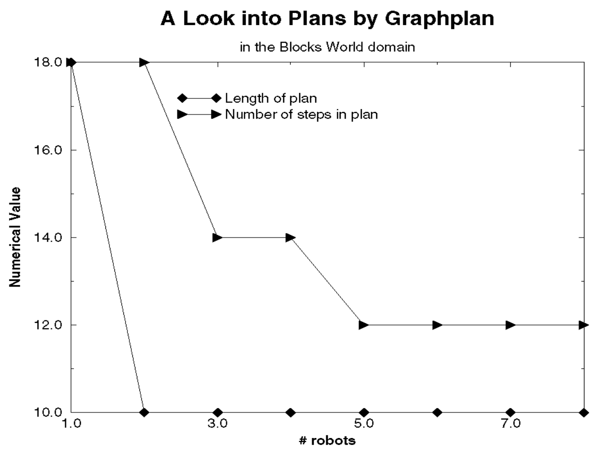
Abstraction

- **Idea**
  - **Abstract some details of the problem or actions.**
  - **Solve the abstracted version.**
  - **Extend the solution to the detailed version**
- **Precondition Abstraction**
  - **Work on satisfying *important* preconditions first**
    - » **Importance judged by:**
      - **Length of plans for subgoals [ABSTRIPS, PABLO]**
      - **Inter-goal relations [ALPINE]**
      - **Distribution-based [HighPoint]**
  - **Strong abstractions (with downward refinement property) are rare**
  - **Effectiveness is planner-dependent**
    - » **Clashes with other heuristics such as *"most constrained first"***

Example: Abstracting Resources

- Most planners thrash by addressing planning and scheduling considerations together
  - Eg. Blocks world, with multiple robot hands
- Idea: Abstract resources away during planning
  - Plan assuming infinite resources
  - Do a post-planning resource allocation phase
  - Re-plan if needed





(with Biplav Srivastava)

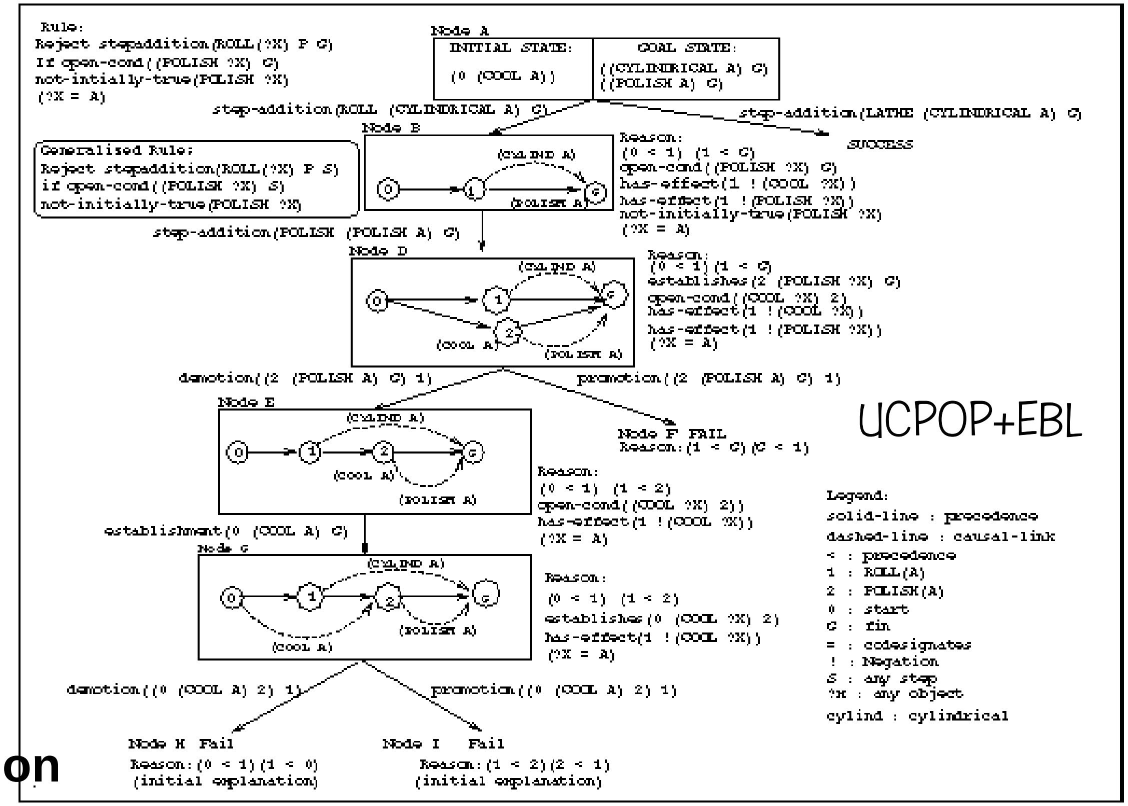
Learning Search Control Rules with EBL

Explain leaf level failures

Regress the explanations to compute interior node failure explanations

Use failure explanations to set up control rules

Problems:
- **-- Most branches end in depth-limits**
  - **>No analytical explanation**
  - **>Use preference rules?**
- **-- THE Utility problem**
  - **>Learn general rules**
  - **>Keep usage statistics & prune useless rules**



If Polished(x)@S &
~Initially-True(Polished(x))
Then
<u>REJECT</u>
Stepadd(Roll(x),Cylindrical(x)@s)

(Kambhampati, Katukam, Qu, 95)

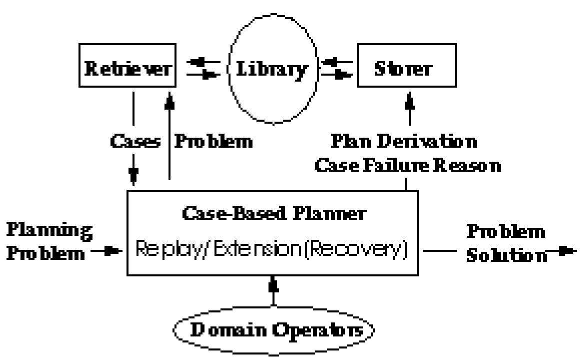
Case-based Planning
Macrops, Reuse, Replay

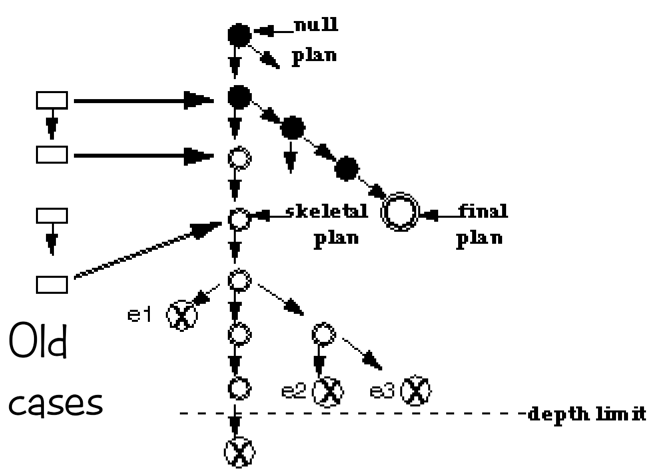
- Structures being reused
  - Opaque vs. Modifiable
  - Solution vs. Solving process (derivation)
- Acquisition of structures to be reused
  - Human given vs. Automatically acquired
- Mechanics of reuse
  - Phased vs. simultaneous
- Costs
  - Storage & Retrieval costs; Solution quality

(Ihrig & Kambhampati, JAIR 97)

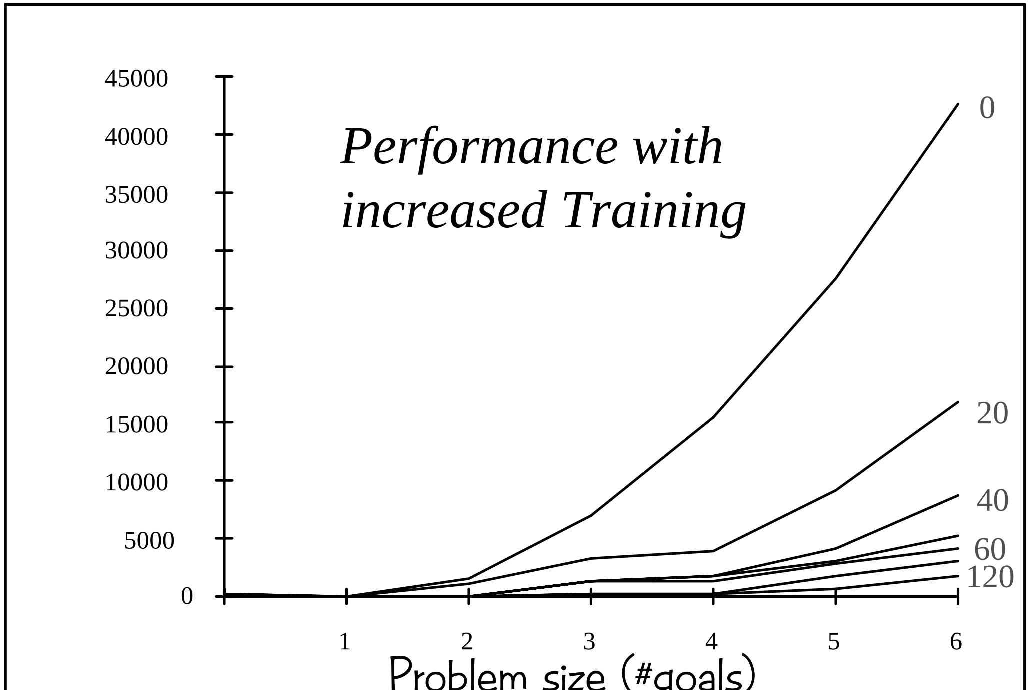
Case-study: DerSNLP

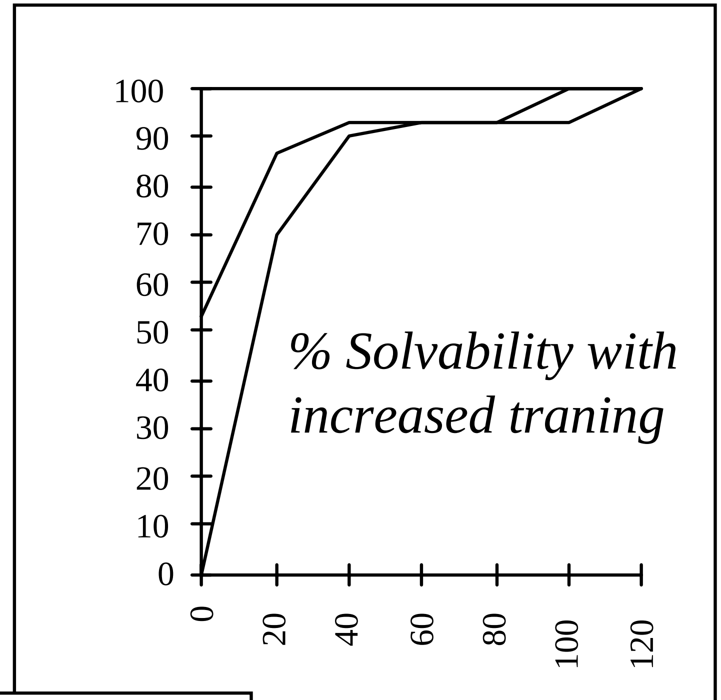
- **Modifiable derivational traces are reused**
- **Traces are automatically acquired during problem solving**
  - **Analyze the interactions among the parts of a plan, and store plans for *non-interacting* subgoals separately**
    - » **Reduces retrieval cost**
  - **Use of EBL failure analysis to detect interactions**
- **All relevant trace fragments are retrieved and replayed before the control is given to from-scratch planner**
  - **Extension failures are traced to individual replayed traces, and their storage indices are modified appropriately**
    - » **Improves retrieval accuracy**

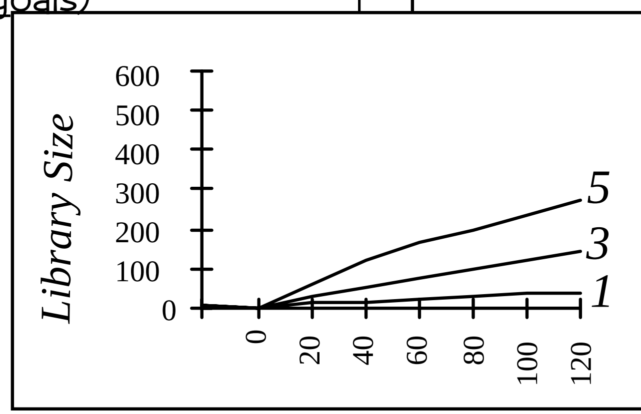




DerSNLP: Results







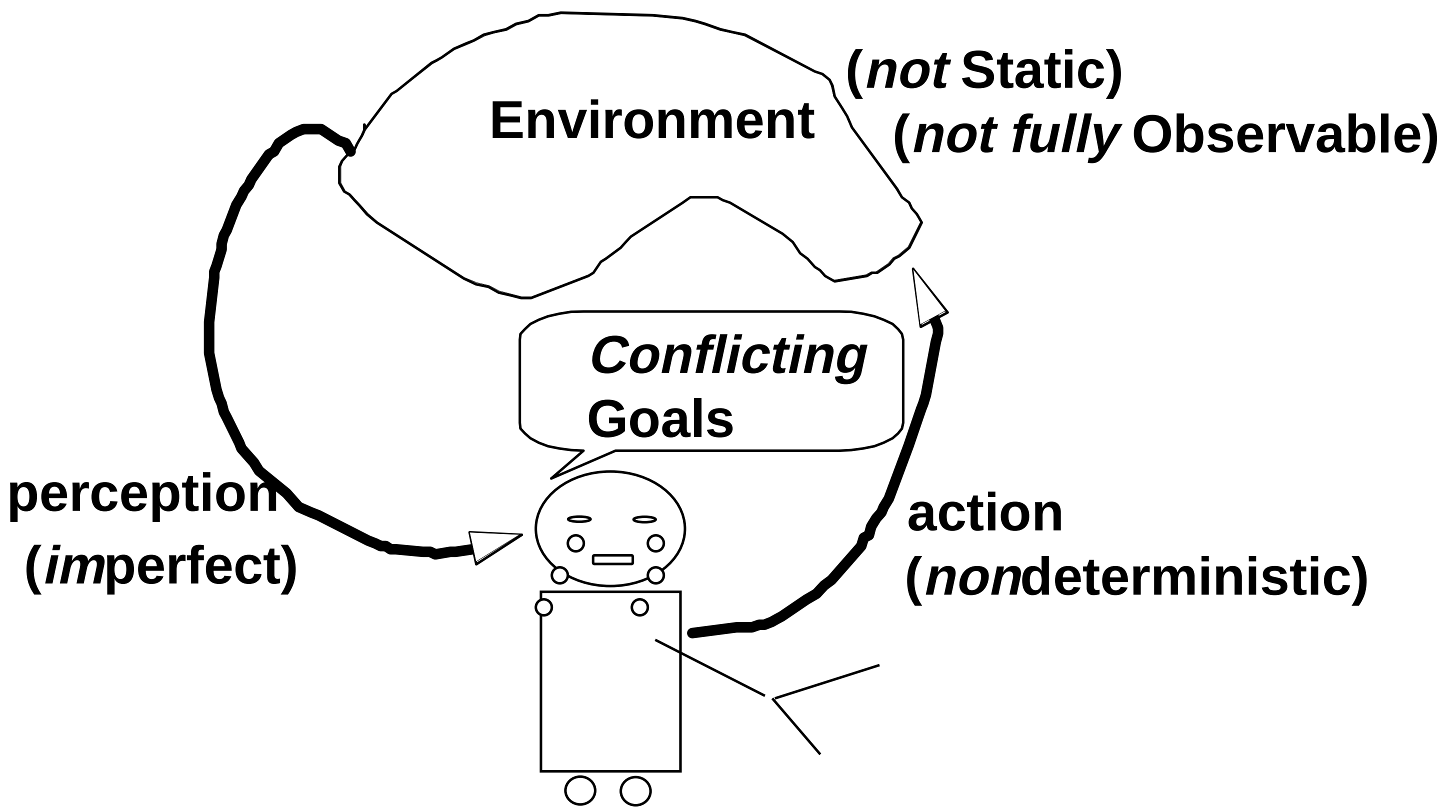
(JAIR, 97)

Reuse in Disjunctive Planning

- Harder to make a disjunctive planner commit to extending a specific plan first
- Options:
  - Support opaque macros along with primitive actions
    - Increases the size of k-step disjunctive plan
    - But a solution may be found at smaller k
  - Modify the problem/domain specification so the old plan's constraints will be respected in any solution (Bailotti et. al.)
  - MAX-SAT formulations of reuse problem
    - Constrain the encoding so that certain steps may have smaller step-action mapping and ordering choices
    - Causal encodings provide better support

[with Amol Mali]

Connections to Non-Classical Planning



The most efficient approaches to all these *ills* are still based on classical planning ideas...

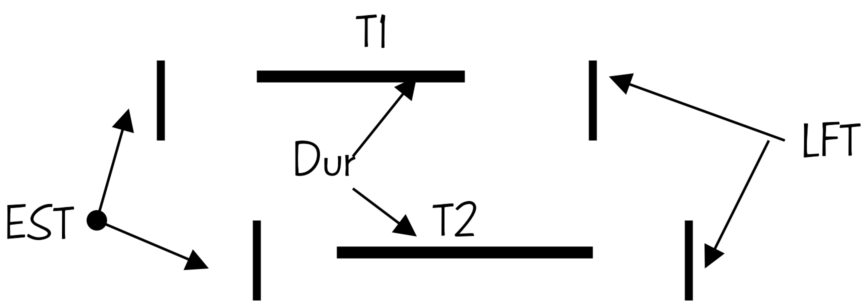
Metric and Temporal constraints

- Problem: Most real-world problems involve actions with durations, goals with deadlines, continuous resources
  - While still being observable, deterministic and static
- APPROACHES:
  - Handling numeric/continous quantities
    - » LPSAT [Wolfman & Weld; 99] integrates a SAT solver with an LP solver
      - ZENO [Penberthy & Weld; AAAI-94] extends UCPOP
    - » ILP encodings [Vossen & Nau; 99; Kautz & Walser; 99]
  - Handling actions with durations
    - » TGP [Smith & Weld; 99] supports actions with durations in Graphplan
  - Integrate planning & scheduling; postpone durations, resources etc. to scheduling phase
    - » [Srivastava & Kambhampati; 99]

Scheduling as CSP

Jobshop scheduling

- Set of jobs
  - » Each job consists of tasks in some (partial) order
- Temporal constraints on jobs
  - » EST, LFT, Duration
- Contention constraints
  - » Each task can be done on a subset of machines



CSP Models

- Tasks as variables
  - » Time points as values
  - » EST, LFT, Machine contention as constraints
- Inter-task precedences as variables

CSP Techniques

- Customized consistency enforcement techniques
  - » ARC-B consistency
- Customized variable/value ordering heuristics
  - » Contention-based
  - » Slack-based
- MaxCSP; B&B searches

Incomplete Information

- **PROBLEM: Values of some state variables are unknown; There are actions capable of sensing (some) of them.**
  - **If k boolean state variables are unknown, then we are in one of 2^k initial states**
  - **Two naïve approaches**
    - » **PLAN/SENSE/EXECUTE : Solve each of the 2^k problems separately; At the execution time sense the appropriate variables, and execute the appropriate plan**
    - » **SENSE/PLAN/EXECUTE: First sense the values of the variables. Solve the problem corresponding to the sensed values**
  - **Problems with naïve approaches**
    - » **Solving the 2^k problems separately is wasteful**
      - **Shared structure (Tree structured plans)**
    - » **Not all variables may be observable (or worth observing)**
      - **Conformant planning**
        - **(Find non-sensing plans that work in all worlds)**
      - **Irrelevant variables (Goal directed planning)**

Incomplete Information: Some Implemented Approaches

- Conjunctive planners
  - CNLP [Peot & Smith; 92] CASSANDRA [Pryor & Collins, 96] add sensing actions to UCPOP; support tree-shaped plans
  - SADL/PUCCINI [Golden & Weld; 96-98] integrates planning and sensing in the context of a UCPOP-like planner
- Disjunctive planners
  - CGP [Smith & Weld, 98] supports conformant planning on Graphplan
  - SGP [Weld et. al., 98] supports conditional planning on Graphplan
    - » One plan-graph per possible world/ Interactions among plangraphs captured through induced mutexes
  - [Rintanen, 99] converts conditional planning to QBF encodings

Dynamic Environments

- **PROBLEM: The world doesn’t sit still. Blind execution of a “correct” plan may not reach goals**
- **APPROACHES:**
  - **PLAN/MONITOR/REPLAN: Monitor the execution; when the observed state differs from the expected one, REPLAN**
    - **Replanning is like reuse except there is added incentive for minimal modification**
      - **Easy to support with conjunctive plan-space planners**
        - **PRIAR [Kambhampati; 92]; DerSNLP [Ihrig & Kambhampati, 97]**
      - **Possible to support with disjunctive causal encodings**
        - **[Mali & Kambhampati]**
  - **MONITOR/REACT/LEARN:**
    - **Policy construction (Universal plans)…**
  - **MODEL OTHER AGENTS CAUSING CHANGE:**
    - **Collaborative/Distributed planning**

Stochastic Actions

- PROBLEM: Action effects are stochastic
  - Actions transform *state-distributions* to state-distributions
  - Maximize "probability" of goal satisfaction
  - Plan assessment itself is hard
- APPROACHES:
  - Conjunctive planners
    - BURIDAN [Hanks et. al., 95] uses UCPOP techniques to put candidate plans together and assesses them
      - *Multiple /redundant supports*
  - Disjunctive planners
    - Pgraphplan [Blum & Langford, 98] modifies Graphplan to support some forms of stochastic planning
      - Forward search; value propagation
    - Maxplan [Majercik & Littman, 98] uses EMAJSAT encodings to solve stochastic planning problems
      - Chance variables & Choice variables. Equivalence classes of models that have the same values of choice variables. Find the equivalence class with maximum probability mass.

Complex & Conflicting Goals

- **Problems & Solutions:**
  - **Goals that have temporal extent (stay alive)**
    - **UCPOP, TLPlan, TGP [Smith& Weld, 99]**
  - **Goals that have mutual conflicts (Sky-dive & Stay Alive)**
  - **Goals that take cost of achievement into account**
  - **Goals that admit degrees of satisfaction (Get rich)**
    - **Branch & Bound approaches; MAXSAT approaches**
      - **Pyrrhus [Williamson & Hanks; 92]**

Decision Theoretic Approaches:
Model goals in terms of factored reward functions
for Markov Decision Processes
--Can utilize tricks and insights from classical planning
[Boutilier, Hanks, Dean; JAIR 99]

Summary

- Refinement planning provides a unifying view
  - Conjunctive Refinement Planners
    - » Effective heuristics
  - Disjunctive Refinement Planners
    - » Refinement
    - » Solution Extraction
      - Direct vs. compilation to CSP/SAT
  - Tradeoffs, Acceleration techniques
- Customization of planners
  - User-assisted
  - Automated
- Related approaches to non-classical planning

Status

- **Exciting times…**
  - **Many approaches with superior scale-up capabilities**
    - » **Broadened views of planning**
  - **Many influences (CSP; OR; MDP; SCM)**
- **Ripe for serious applications**
  - **VICAR [JPL]; DeepSpace monitoring [NASA/AMES]**
  - **Plant monitoring [Ayslett et. al.]**
  - **Manufacturing Process Planning [Nau et. al.; Kambhampati et. al]**
  - **Supply chain management/ Logistics**
    - » **Industrial "Planning" does not have to the optimal scheduling of an inoptimal action selection!**

Resources

- Mailing Lists
  - Planning list digest
    - » http://rakaposhi.eas.asu.edu/planning-list-digest
  - U.K. P & S List
    - » http://www.salford.ac.uk/planning/home.html
- Special Conferences
  - Intl. Conf. on AI Planning & Scheduling
    - » http://www.isi.edu/aips (April 2000, Breckenrdige, CO)
  - European Conference on Planning
    - » http://www.informatik.uni-ulm.de/ki/ecp-99.html
  - Also, AAAI, IJCAI, ECAI, Spring/Fall Symposia
- Courses
  - ASU Planning Seminar Online Notes (1999, 1997, 1995, 1993)
    - » http://rakaposhi.eas.asu.edu/planning-class.html