united nations
educational, scientific
and cultural
organization

international atomic
energy agency

the
abdus salam
international centre for theoretical physics

SMR/1423 - 22

CONFERENCE ON
"TYPICAL-CASE COMPLEXITY, RANDOMNESS AND ANALYSIS OF SEARCH ALGORITHMS"

(5 - 7 September 2002)

" Planning problems "

presented by:

E. Giunchiglia
Università di Genova

An Introduction to AI Planning

Enrico Giunchiglia
DIST – Università di Genova

AI Planning: an example

Suppose that you have to go to the airport.

Which actions do you have to perform in order to achieve your goal starting from here?

To answer this question, a computer has to know the "initial state", the available "actions", and the "goal".

Planning: an (informal) definition

Given 2 disjoint sets of variables:

1. *Fluents*, for describing the states of the world, and
2. *Actions,* for characterizing transitions between states,

3. A planning problem is a triple <I,D,G> where:

- I describes the initial state of affairs
- D is the domain and characterizes how each set of actions affects the world
- G is the goal, i.e., the condition to achieve.

The problem is: does there exist a combination of actions that, if executed from I, achieves the goal G?

Restrictions

Here we restrict to approaches in which:

- A state is a Boolean interpretation of the fluent signature,
- Actions corresponds to transitions between states,
- I and G are subsets of the set of states,
- A plan is a sequence of actions
- The goal is achieved if *each* transition starting from I and which follows the plan ends up in a state in G.

The state transistion model

Thus, a domain D corresponds to a transition system:

- the states S are the interpretations of the fluent signature,
- Each actions is a mapping from $S \rightarrow Pow(S)$.

We say that an action A is

- *Executable in a state s,* if $|A(s)| \geq 1$,
- *Deterministic,* if for each state s, $|A(s)| \leq 1$.

Outline

- Classical Planning:
  - Representing Actions (the transitions):
    - STRIPS
    - ADL (Action Description Language)
  - Planning systems:
    - Via SAT encodings (Medic, Blackbox)
    - Via Heuristic search (HSP, FF)

- Beyond Classical Planning:
  - Representing Actions (the transitions):
    - *C* Language
  - Planning systems:
    - *Ccalc, C*-plan

What is not covered

- Classical planning:
  - Situation Calculus representations and procedures
  - Planning in the space of plans (POP)
  - Planning with planning graphs
  - Advanced topics

- Beyond Classical planning:
  - First order extensions (e.g., for resources)
  - Extended Goals
  - Planning with control information
  - ……..

Classical Planning

Classical Planning

- Domains are specified using a first-order language.
- More specifically, a planning domain is defined by a set of operators which are a parameterized representation of the transitions available in the domain.

Assumptions

- Completely specified initial state.
- Deterministic, completely specified actions, which do not modify the set of objects in the world.
- The set of objects is finite and given.
- The goal is a property of an individual state.

Representing the Initial state

- Simple definition: The initial state is maximally consistent set of ground literals.
- Used definition: The initial state is a set of ground atoms.

Notice that:

- Domain closure is implicit.
  - The set of constants mentioned in the set are all the objects in the world.
- The unique names assumption is implicit.
  - Distinct constants are not equal.

(Like a database)

Representing the Initial state

Independently from the representation:

- it is possible to efficiently determine in the initial state the truth of any first-order sentence.
- More generally, given any first-order formula (with free variables) it is possible to determine the set of instantiations of these variables (with constants) that satisfy the formula in the initial state.

Representing the Initial state

However, in practice, representation matters:

- the relational representation can be much more compact than the propositional representation.
- E.g., in the standard blocks world with 500 blocks in the initial state, there are about 250,000 possible on(x,y) relations.
  - 25K byte bit vector (40,000 states = 1GB)
  - A block can only be on one other block so only 500 possible on(x,y) relations in a database (40,000 states = 20MB).

Representing the Goal

- Representation of the goal
  - A set of ground literals
  - A state satisfies the goal if it satisfies all literals in the goal
  - Other possibilities
    - A more complex condition on a state, specified with a first-order formula.
    - A condition on the sequence of states visited by the plan (a "Temporally Extended Goal")
    - The difficulty here lies in creating methods for effectively searching for plans satisfying these more complex goals.

Representation of Operators

- Operators specify the possible state transitions for a planning domain.
- For any particular planning problem, it is necessary to use the initial state as well as the operator specification to determine the corresponding transition system.
- To be problem independent operators use parameters.

Representation of Operators

- Since operators are domain specific, problem independent
  - Possible to develop methods that compute properties of transitions that apply to all possible problems in the domain.
  - The representation is more compact, independent of the size of the particular planning problem.

STRIPS Representation

- STRIPS is the simplest and the second oldest representation of operators in AI.
- When that the initial state is represented by a database of positive facts, STRIPS can be viewed as being simply a way of specifying an update to this database.

STRIPS Representation

```
(def-strips-operator (pickup ?x)
    (pre (handempty) (clear ?x) (ontable ?x))
    (add (holding ?x))
    (del (handempty) (clear ?x) (ontable ?x)))
```

STRIPS Representation

(def-strips-operator (pickup ?x)

operator name
and parameters

(pre (handempty) (clear ?x) (ontable ?x))

(add (holding ?x))

(del (handempty) (clear ?x) (ontable ?x)))

STRIPS Representation

(def-strips-operator (pickup ?x)

(pre (handempty) (clear ?x) (ontable ?x))

List of predicates that must hold in the current state for the action to be applicable

(add (holding ?x))
(del (handempty) (clear ?x) (ontable ?x)))

STRIPS Representation

(def-strips-operator (pickup ?x)

(pre (handempty) (clear ?x) (ontable ?x))

(add (holding ?x))

List of predicates that must be true in the next state

(del (handempty) (clear ?x) (ontable ?x)))

STRIPS Representation

(def-strips-operator (pickup ?x)

(pre (handempty) (clear ?x) (ontable ?x))

(add (holding ?x))

(del (handempty) (clear ?x) (ontable ?x)))

List of predicates that must be false in the next state

STRIPS Representation

- Given the initial state
  - All instantiations of the parameter ?x that satisfy the precondition

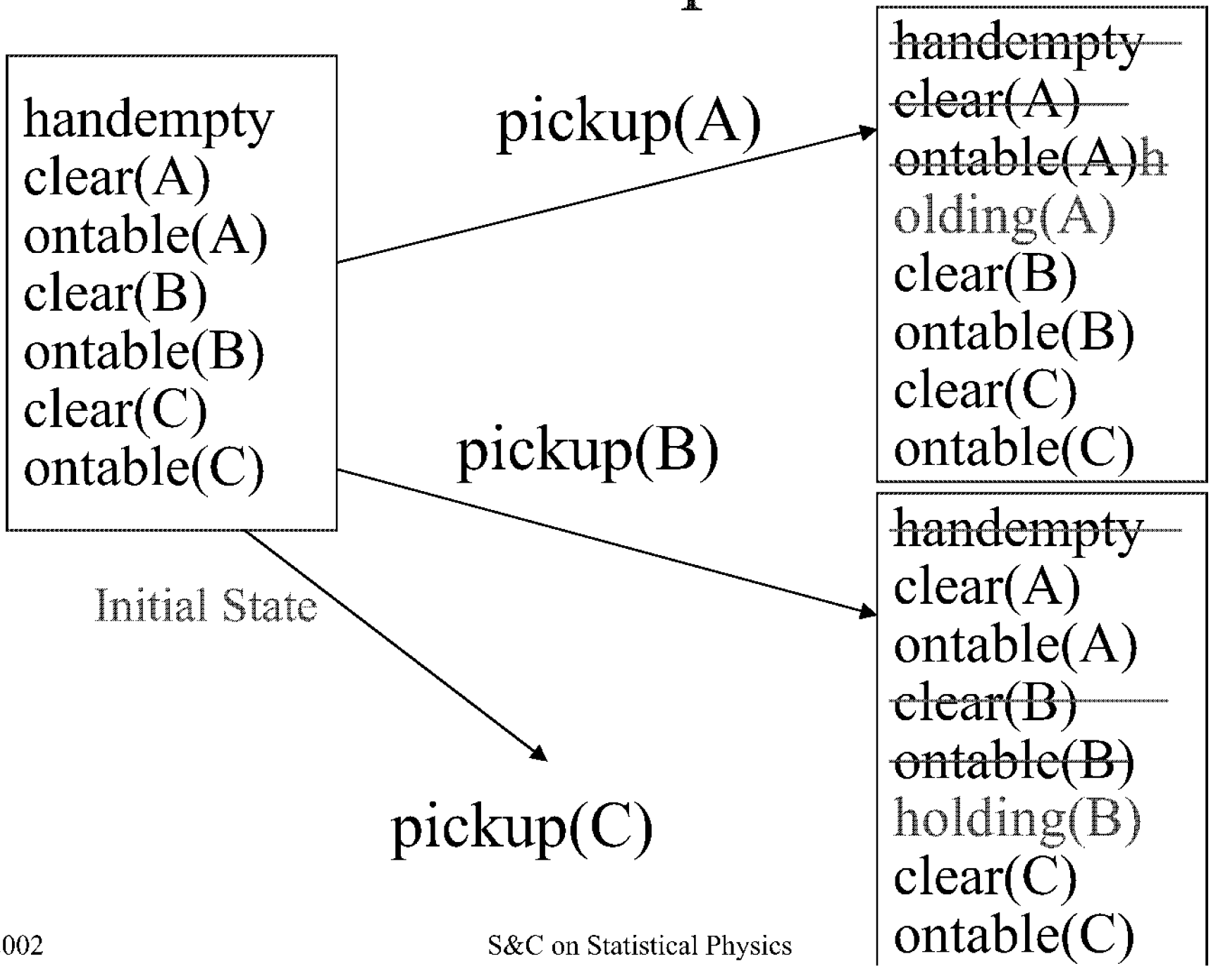
    (and (handempty) (clear ?x) (ontable ?x))

    produce a different action (transition) that can be applied to the initial state.
  - Actions whose preconditions are not satisfied are not legal transitions.

STRIPS Representation

- Actions are deterministic:
  - Given a particular instantiation of the parameters, the action specifies a finite collection of ground atomic formulas that must be made true and another collection that must be made false in the successor state.
- Nothing else is changed! (Frame assumption). This has many algorithmic consequences.

STRIPS Representation



STRIPS Representation

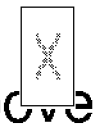
- The properties of the initial state and the operators imply that from a finite collection of operators it is possible to determine
  - the finite collection of actions that can be applied to the initial state.
  - In each successor state generated by these actions, we can once again evaluate all logical formulas, and thus once again determine the set of all applicable actions.
- Hence, we can incrementally generate the set of all states reachable from the initial state by sequences of actions.
- This is the forward space, and we can search for plans in this space. (To be examined later).

ADL Representation

- Action description language due to Pednault.
- Generalizes STRIPS to allow for
  - Arbitrary first-order preconditions
  - Conditional effects
  - Universal effects
  - Functions

ADL Representation

```
(def-adl-operator (move ?x ?old ?new)
  (pre (and (on ?x ?old) (not (?old = ?new))
            (not (exists (?z) (on ?z ?x)))
            (not (exists (?z) (on ?z ?new)))))
  (add (on ?x ?new))
  (del (on ?x ?old))
  (forall (?z)
    (implies (above ?x ?z) (del (above ?x ?z))))
  (forall (?z)
    (implies (above ?new ?z) (add (above ?x ?z)))))
```



ADL Representation

```
(def-adl-operator (move ?x ?old ?new)
  (pre (and (on ?x ?old) (not (?old = ?new))
            (not (exists (?z) (on ?z ?x)))
            (not (exists (?z) (on ?z ?new)))))
```

First-order preconditions.

```
  (add (on ?x ?new))
  (del (on ?x ?old))
  (forall (?z)
    (implies (above ?x ?z) (del (above ?x ?z))))
  (forall (?z)
    (implies (above ?new ?z) (add (above ?x ?z)))))
```

ADL Representation

```
(def-adl-operator (move ?x ?old ?new)
  (pre (and (on ?x ?old) (not (?old = ?new))
            (not (exists (?z) (on ?z ?x)))
            (not (exists (?z) (on ?z ?new)))))
  (add (on ?x ?new))
  (del (on ?x ?old))
  (forall (?z)
    (implies (above ?x ?z) (del (above ?x ?z))))
  (forall (?z)
    (implies (above ?new ?z) (add (above ?x ?z)))))
```

Conditional effects where quantification can be used to specify the set of atomic updates.

ADL Representation

- As in STRIPS
  - Every action specifies a finite collection of ground atomic formulas that must be made true and another collection that must be made false in the successor state
  - Nothing else changes.
- Given the completeness properties of the initial state
  - it is still possible to compute all applicable actions, and the effects of these actions.
  - All successors states have the same completeness properties.

ADL Representation

- So, it is remains possible to generate and search the forward space with ADL actions.
- ADL actions do pose some additional complexities for alternate search spaces.
  - One approach is to compile all ADL actions into a set of STRIPS actions.
    - Can yield an exponential number of STRIPS actions [Nebel, 2000]
  - An alternative is to develop techniques for dealing directly with ADL actions (perhaps with some restrictions) in these alternate search spaces
    - UCPOP, ADL for searching the space of partially ordered plans. [Penberthy & Weld, 1992]

Features of Action Representations

- Actions cause modular updates, they affect only a (generally) small set of predicates and a small set objects.
- The frame assumption (that most things are unchanged) is built into these representations.
- Specified in a parameterized manner.
- (Relatively) easy to compute the set of executable actions, and the corresponding resulting states.
- These features all play a role in the search techniques developed in planning.

Searching for Plans

- We have seen that it is possible to search for plans in the space of states.
- There are other types of spaces over which plans can be searched for.
- Much of the work in planning has been devoted to developing methods for searching such alternate spaces.
- Now we will describe some of the spaces that can be searched for plans.

Propositional Spaces

- Under domain closure all possible states and all possible actions can be represented by a collection of propositions: each possible instantiation of the predicates and each possible instantiation of the operators.
- In order to represent the search space with propositions, we simply need to impose a fixed bound on plan length.
- Since the length of the plan is unknown, we can incrementally increase the bound on length, at each state doing search in the resulting propositional space.

Propositional Spaces

State at T0	Action at T0	State at T1	
on(A,B,0)	pickup(A,0)	on(A,B,1)	
on(B,A,0)	pickup(B,0)	on(B,A,1)	
onTable(A,0)	putdown(A,0)	on(A,Table,1)	
onTable(B,0)	putdown(B,0)	on(B,Table,1)	
clear(A,0)	stack(A,B,0)	clear(A,1)	
clear(B,0)	stack(B,A,0)	clear(B,1)	...
handempty(0)		handempty(1)	
holding(A,0)		holding(A,1)	
holding(B,0)		holding(B,1)	
			...

Simplest idea—a set of propositions to specify each of the k-states and k-actions taken in the k-step plan.

Now specify the conditions required to make this a k-step plan.

Propositional Spaces

on(A,B,0)
on(B,A,0)
onTable(A,0)
onTable(B,0)
clear(A,0)
clear(B,0)
handempty(0)
holding(A,0)
holding(B,0)

pickup(A,0)
pickup(B,0)
putdown(A,0)
putdown(B,0)
stack(A,B,0)
stack(B,A,0)

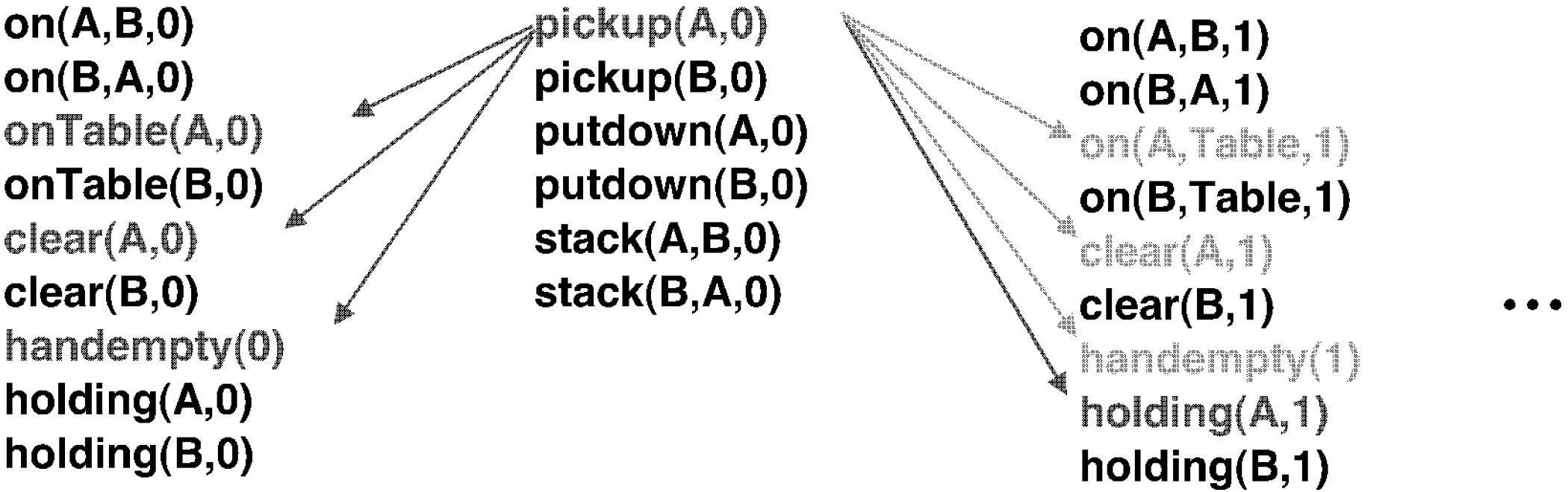
on(A,B,1)
on(B,A,1)
on(A,Table,1)
on(B,Table,1)
clear(A,1)
clear(B,1)
handempty(1)
holding(A,1)
holding(B,1)

• • •

State at T0

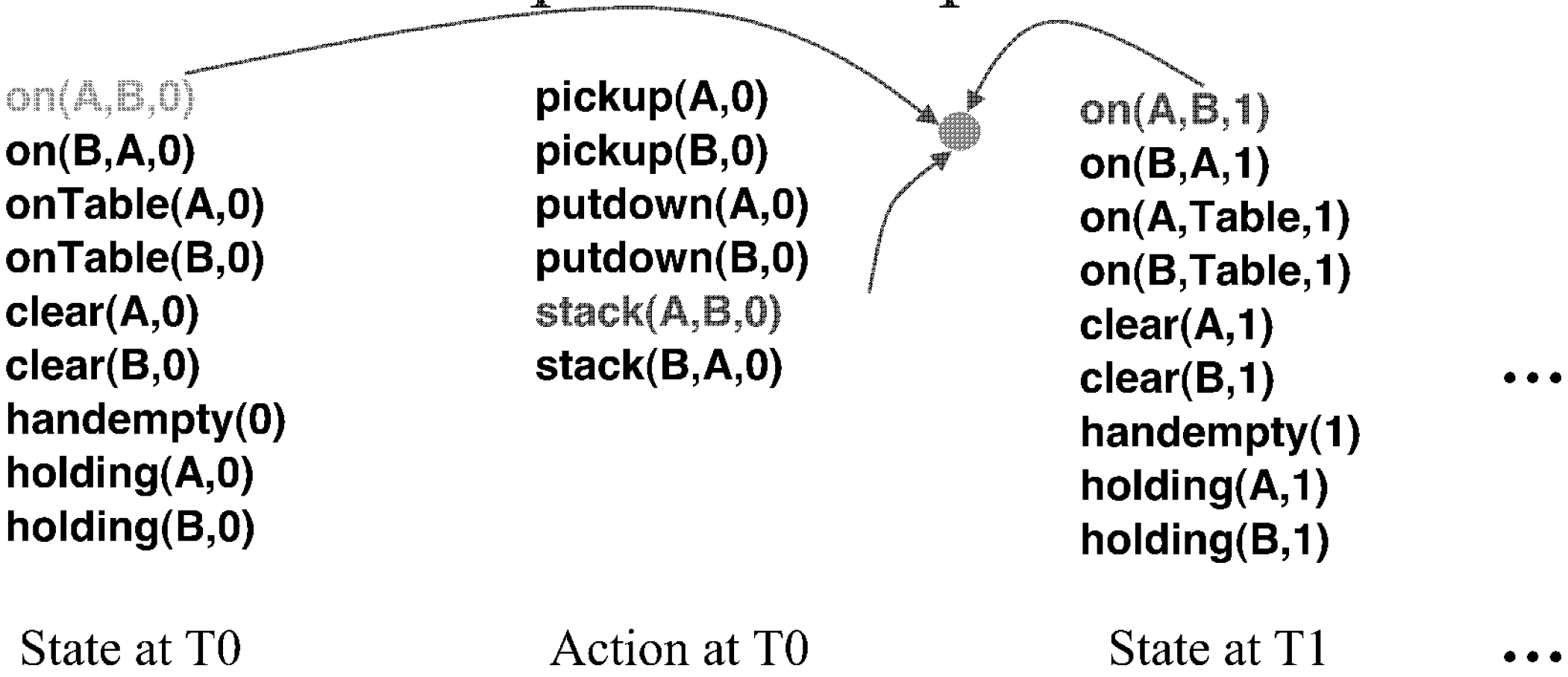
Initial state forces some propositions to be true and the others false. The goal forces some propositions at step k to be true.

Propositional Spaces



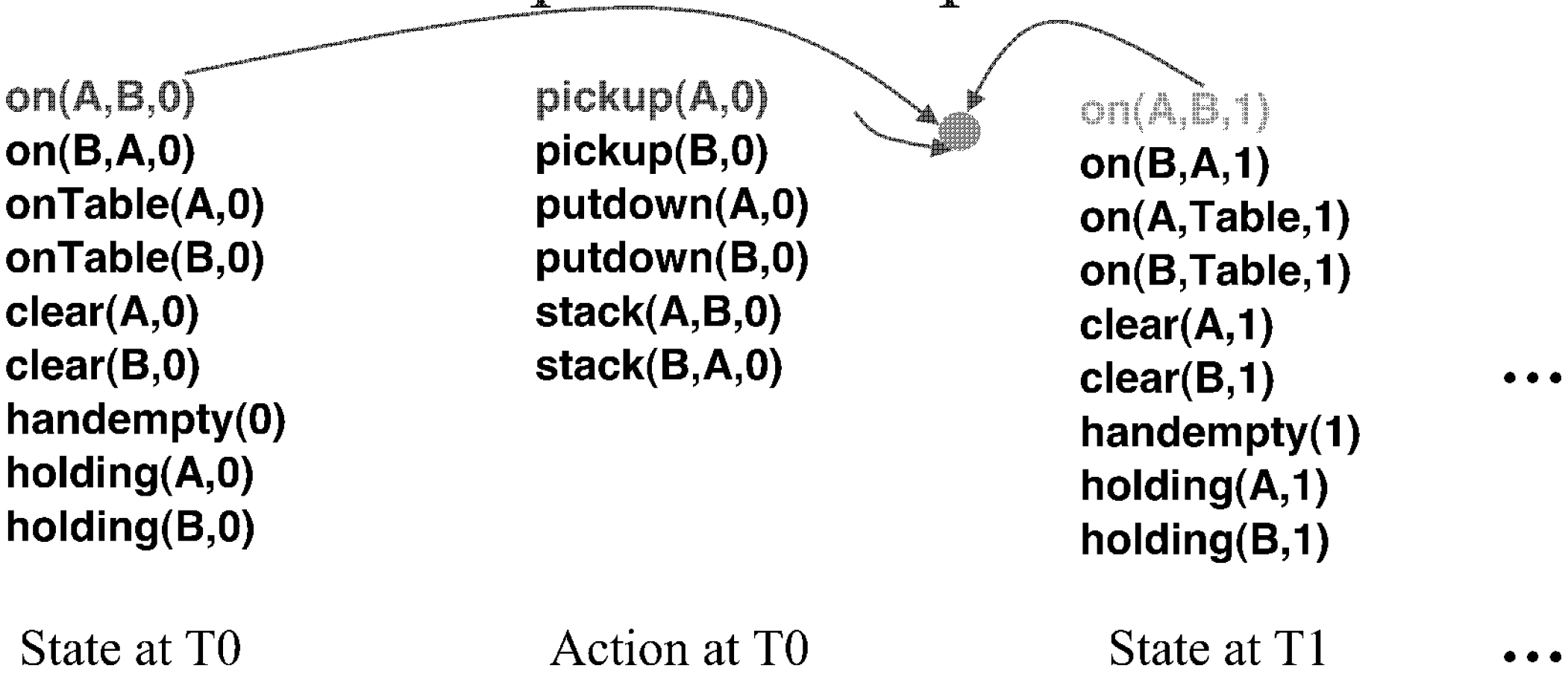
If an action is true, its preconditions must be true and its add effects must be true its delete effects must be false. (Easy for STRIPS, harder for ADL)

Propositional Spaces



Frame assumption: a proposition cannot change its value unless it is changed by an action.

Propositional Spaces



Frame assumption: a proposition cannot change its value unless it is changed by an action.

Propositional Spaces

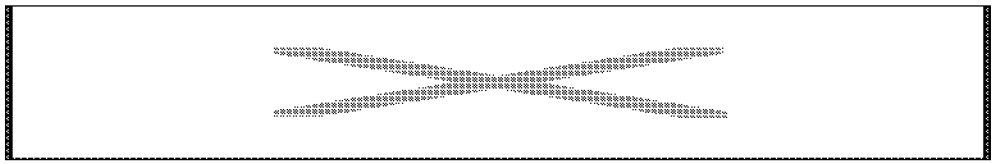
Several encodings of the transition relation are possible in propositional logic, each characterized by

1. The kind of frame axioms used (classical, explanatory)
2. The specific action representation (regular, split, bitwise)
3. The mutual exclusion axioms enforced between actions (sequential, parallel)
4. (Ernst, Millstein, Weld, 1997)

5. Let $TR(i,i+1)$ be an encoding of the transition relation between time i and $i+1$

Propositional Spaces

- There is a one-to-one correspondence between plans of length *n* and assignment satisfying



(Kautz, Selman, 1992). Notice similarity with "Bounded Model Checking" (Biere, et al. 1998) in FV.

- Then we solve it with standard SAT solvers: the state of the action variables at each time step in the solution specifies the plan.
  - Fast for smaller problems, but the size of the SAT problem grows as a high order polynomial.
  - $A = O(|Ops||Dom|^{Arity(Ops)})$ --- number of actions
  - need $O(nA^2)$ clauses [Kautz, McAllester, Selman, 1996]
  - E.g. binary actions like *stack(x,y)*, $O(|Dom|^4)$
  - A 19 block problem in the blocks world generates over 17,000,000 clauses and 40,000 variables.

Forward Space

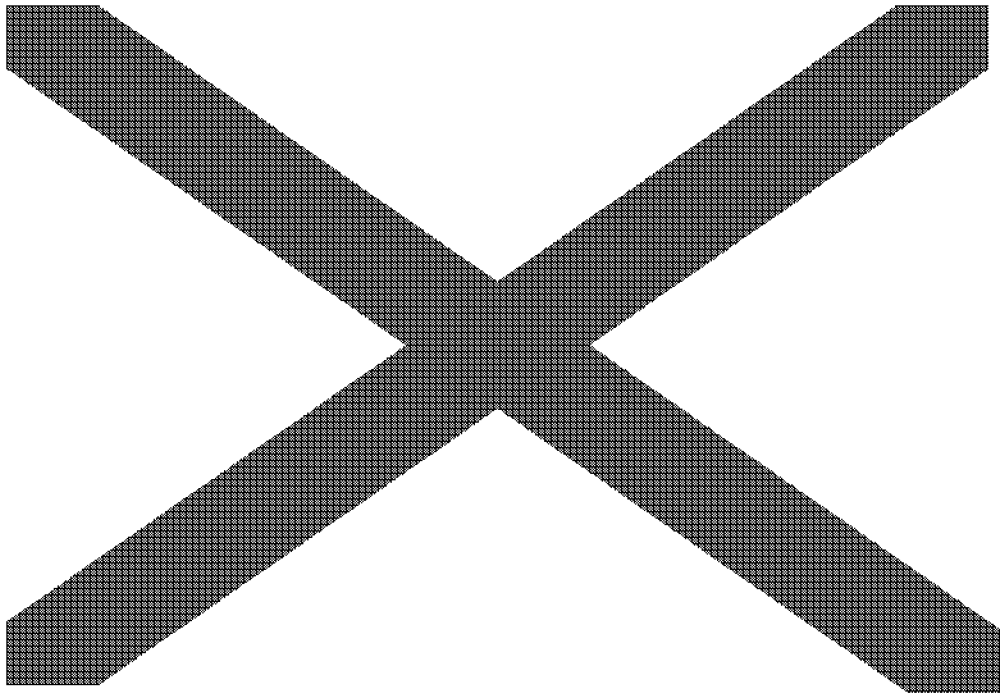
- The conceptually simplest search space is simply to directly apply actions to the initial state, searching for a state that satisfies the goal.
- The difficult has been guiding the search towards goal states.
- A classical technique in AI is to compute a heuristic estimate of the distance to the goal.
- One guides search by first exploring those states that appear to be closer to the goal.

Distance Heuristics

- A standard method for generating heuristics is to consider a relaxed version of the problem, measuring the distance to the goal in this relaxed version.
- A common relaxation in planning is to consider a problem in which the delete effects of all actions are removed.
- Give a set of actions without deletes, it becomes possible to find a plan in polynomial time (if one exists).
  - It remains hard (NP-complete) to find a plan with the minimal number of actions.
  - [Hoffman & Nebel 2001]

Distance Heursitics

- FF uses this relaxation and distance heuristics are computed from Graphplan like reachability analysis.
- The estimate is not ensured to be a lower bound of the actual distance, and so the returned plan is not guaranteed to be optimal.
- In the IPC2, this idea was the fastest and most scalable methods for finding plans.
- Other heuristics including admissible heuristics, have been tried (see, e.g., [Geffner & Haslum 2000]).



Beyond Classical Planning

Beyond Classical planning

Classical planning makes some "simplifying" assumptions, such as:

- The initial state is completely specified
- The domain is deterministic
- The Goal is a property of states

Which are not always ensured to hold in many cases.

Beyond Classical planning

Representation languages for classical planning have built-in some other "simplifying" assumptions, such as:

- The state of the world changes only because of actions
- No concurrency

Further, it allows only for expressing actions' effects.

Beyond Classical planning

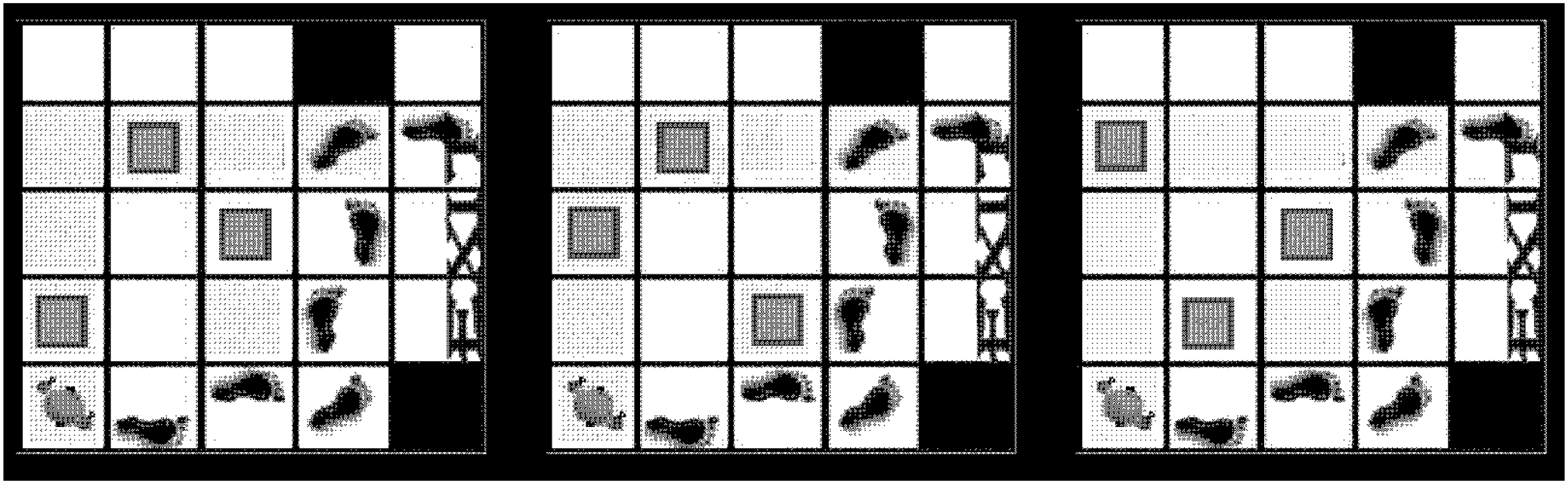
To overcome these problems, several more expressive representation languages have been proposed.

Here we briefly see action language *C* (Giunchiglia, Lifschitz, 1998):

- Very expressive (concurrency, constraints, nondeterminism, enviromental changes, …)
- Supported by ccalc and *C*-plan

A simple example

- Red boxes are obstacles, whose location is only partially known. Black boxes are fixed obstacles.



- Robots have to reach the exit without hitting obstacles.

Several possible definition of "Plan"

- Plan as a sequence of actions (*conformant planning*)
- Plan as a sequence of conditional statements (*contingent planning*)
- Plan as a sequence of conditional statements with loops

ccalc: language specification

```
:- constants
  r1, r2                          :: robot;
  1..5                            :: location;
  at(robot,location,location)  :: inertialFluent;
  occ(location,location)          :: inertialFluent;
  north(robot)                    :: action;
  east(robot)                     :: action;
  west(robot)                     :: action;
  south(robot)                    :: action.
```

ccalc: problem specification

```
:- plan
facts ::
  0: at(r1,1,1),
  0:  (/\)X: (\/)Y: occ(X,Y),
  0:  (/\)L: (/\)J: (/\)K:
          (J<K ->> (-occ(J,L) ++ -occ(K,L))),
  0: ( occ(1,4) && occ(2,2) && occ(4,3) );
goals ::
  1: (\/)Y: at(r1,5,Y).
```

ccalc: domain specification
Constraints

% no robot may be where an object is
never at(R,X,Y) && occ(X,Y).

% every robot has to be somewhere
always (\/)X: (\/)Y: at(R,X,Y).

% a robot can't be at two places at the same time
caused -at(R,X,Y) if at(R,X1,Y1) && -((X is X1) && (Y is Y1)).

ccalc: domain specification
Preconditions of actions

```
% no robot may move outside the grid boundaries
nonexecutable north(R) if at(R,X,5).
nonexecutable east(R)  if at(R,5,Y).
nonexecutable west(R)  if at(R,1,Y).
nonexecutable south(R) if at(R,X,1).

% no robot may go in two directions at the same time
nonexecutable north(R) && east(R).
nonexecutable north(R) && south(R).
nonexecutable north(R) && west(R).
nonexecutable east(R)  && south(R).
nonexecutable east(R)  && west(R).
nonexecutable south(R) && west(R).
```

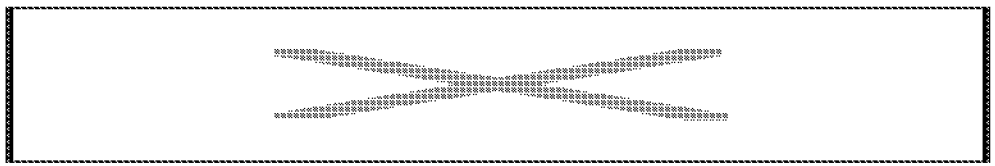
ccalc: domain specification
Effects of actions

```
% what happens if a robot moves?
north(R) causes at(R,X,Y1) if at(R,X,Y) && Y1 is Y+1.
east(R)  causes at(R,X1,Y) if at(R,X,Y) && X1 is X+1.
south(R) causes at(R,X,Y1) if at(R,X,Y) && Y1 is Y-1.
west(R)  causes at(R,X1,Y) if at(R,X,Y) && X1 is X-1.
```

Planning in ccalc

- It is possible to compute a propositional formula whose satisfying assignments are one-to-one with the transitions of the domains.
- If the domain is deterministic, plans of length *n* correspond to assignments satisfying

as in th



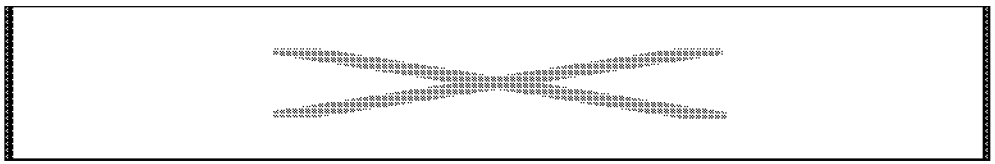
Planning in *C-plan*

- If the domain is not deterministic, it is still possible to use a SAT-based approach (Giunchiglia, 2000) in two steps:

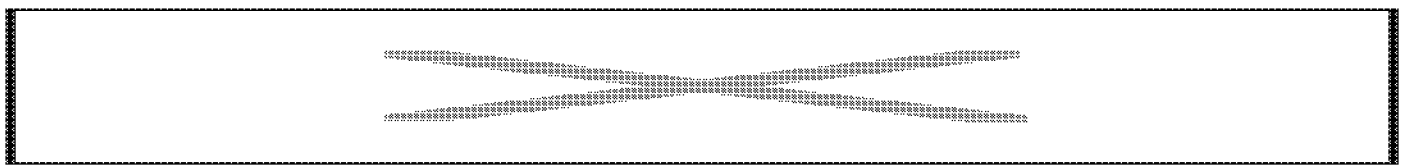
1. Find *possible* plans, i.e., plans that reach the goal for *some* possible configuration of the obstacles.
2. Check (under suitable conditions) if a "possible" plan *A1;A2;...;An* is *valid,* i.e., if it guaranteed to reach the goal for *all* the possible configurations of the obstacles.

Planning in *C-plan*

A *possible* plan of length *n* corresponds to an assignment satisfying



(Under suitable conditions), a "possible" plan *A1;A2;...;An* is *valid* if

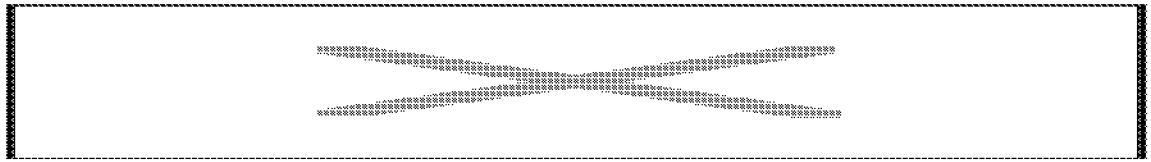


Planning in *C-plan*

These two tests can be implemented using two nested SAT-procedures, or can be encoded as a QBF formula.

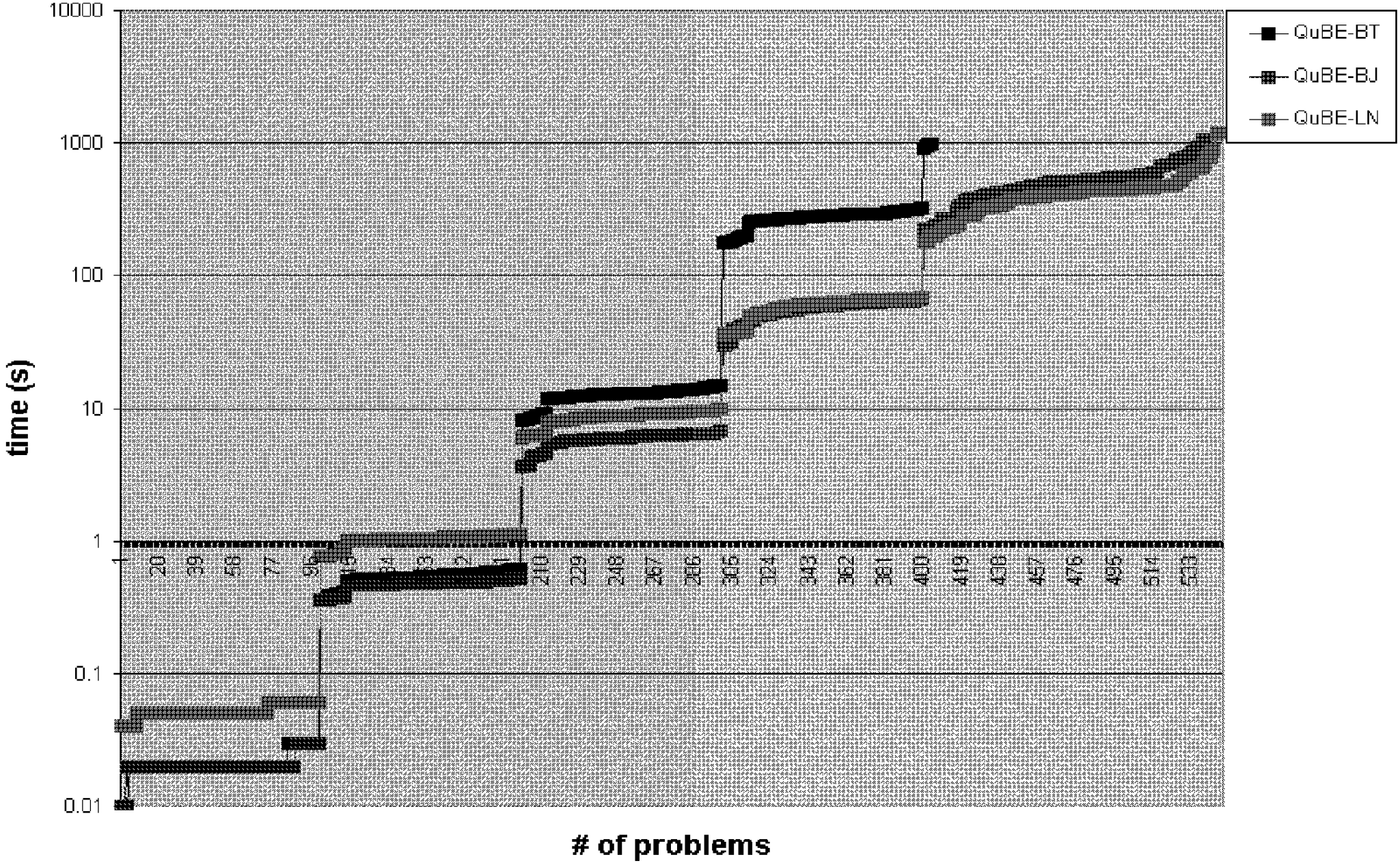
In the first approach, several optimizations are possible to reduce the number of possible plans generated and then tested.

Planning in *C-plan*



function C-sat() return C-sat gendp(cnf(P), {}).

function C-sat gendp(φ, μ)
    if φ = {} then return C-sat test(μ);
    if {} $\in \varphi$ then return False;
    Unit-propagation(φ, μ);
    L := { a literal occurring in φ };
    return C-sat gendp(assign(L, φ),$\mu \cup$ {L}) or
            C-sat gendp(assign(-L, φ),$\mu \cup$ {¬L}).


10000
1000
100
10
1
0.1
0.01
time (s)
QuBE-BT
QuBE-BJ
QuBE-LN
# of problems

Conclusions

- AI planning uses
  - Relational representations.
  - Several representation languages with different expressive capabilites.
  - Several planning procedures, especially for STRIPS and ADL.
  - Planning graphs and forward search are very effective for STRIPS and ADL.
  - Planning as satisfiability can be used with any representation language.
  - There are several extensions to the notions here introduced.