Automated Planning

Planning by Translation to Propositional Satisfiability

Jonas Kvarnström
Automated Planning Group
Department of Computer and Information Science
Linköping University

jonas.kvarnstrom@liu.se – 2017

Background

jonkv@ida

- Propositional satisfiability (SAT):
  - Let φ be a propositional formula
    - For example, $(\text{walking} \rightarrow \text{alive}) \wedge (\text{alive} \rightarrow \text{breathing}) \wedge (\text{breathing})$
  - Is there a **SAT solution**:
    An **assignment** of truth values to all propositions that **satisfies** φ?

Assignments: 8

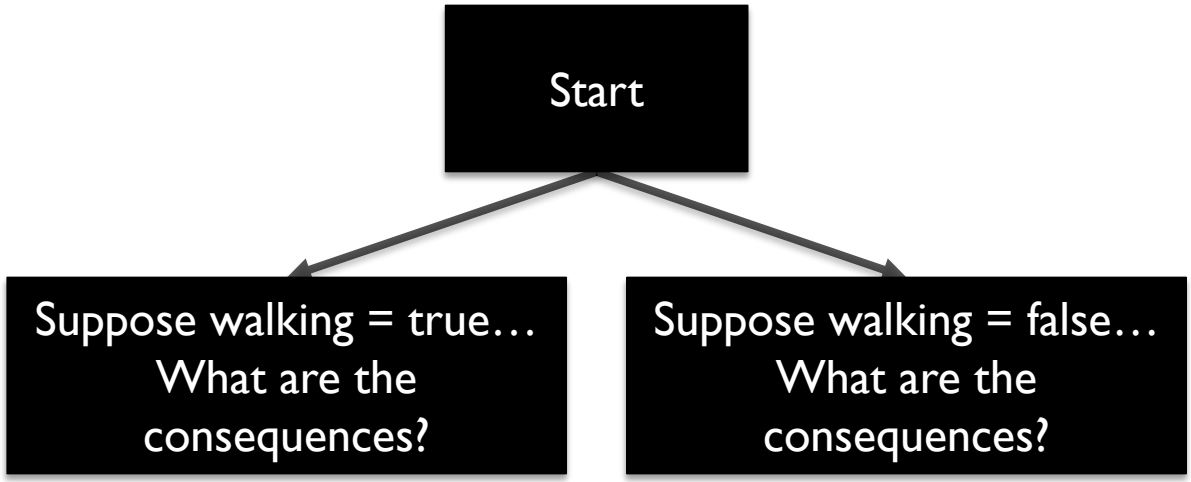
walking	alive	breathing	φ
–	–	–	–
–	–	true	true
–	true	–	–
–	true	true	true
true	–	–	–
true	–	true	–
true	true	–	–
true	true	true	true

Solutions: 3

Background (2)

jonkv@ida

- SAT: The first problem ever proven **NP-complete**!
  - A great deal of research in **efficient algorithms**
    (exponential in the worst case,
    but efficient for many "real" problems)
  - Generally uses **search methods**



Let's try to **translate** planning problems into SAT!

Make use of all these efficient algorithms…

Running Example

jonkv@ida

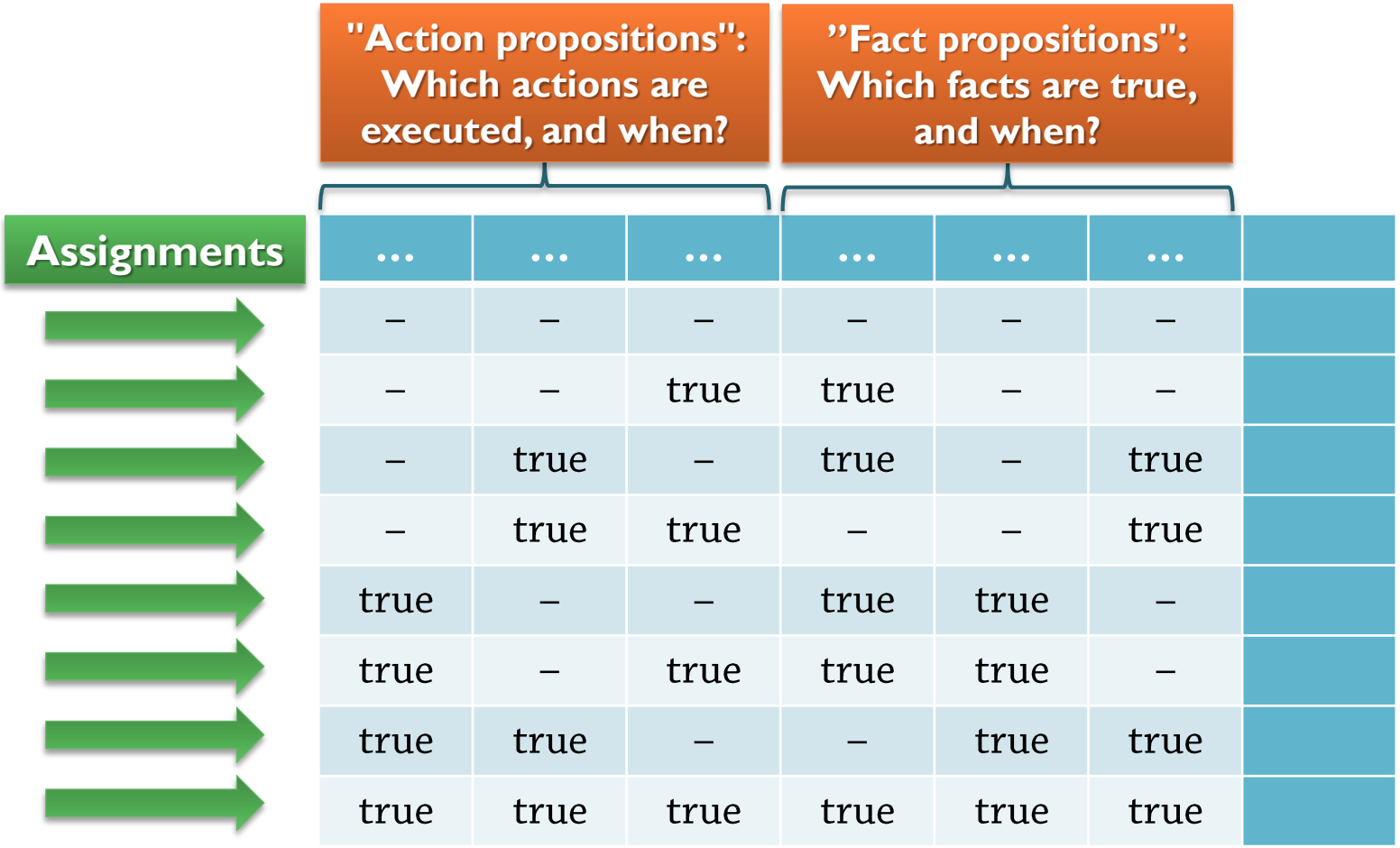
- Very simple **planning domain**
  - Types: – robot and box are subtypes of object
    – location
  - Two predicates: – at(object o, location l)
    – carrying(robot r, box b)
  - First operator: **move**(robot r, location from, location t
    - precond: at(r, from)
    - effects: at(r, to) ^ ¬ at(r, from)
  - Second operator: **pickup**(robot r, box b, location l)
    - precond: at(r, l) ^ at(b, l)
    - effects: carrying(r, b) ^ ¬ at(b, l)
- Corresponding **problem instance**:
  - one robot: rob1
    one box: box1
    two locations: loc1, loc2

Key Ideas

Key Ideas (1)

jonkv@ida

- Key idea: Each **SAT assignment** should correspond to…
  - A specific action sequence
  - A specific state sequence



…	…	…	…	…	…	
–	–	–	–	–	–	
–	–	true	true	–	–	
–	true	–	true	–	true	
–	true	true	–	–	true	
true	–	–	true	true	–	
true	–	true	true	true	–	
true	true	–	–	true	true	
true	true	true	true	true	true	

Key Ideas (2)

jonkv@ida

- Each **SAT solution** should correspond to a **solution plan**
  - Requires a very **complex** formula φ related to initial state, goal, actions, …

"Action propositions": Which actions are executed, and when?			"Fact propositions": Which facts are true, and when?				
…	…	…	…	…	…	φ	Solutions
–	–	–	–	–	–	–	
–	–	true	true	–	–	–	
–	true	–	true	–	true	–	
–	true	true	–	–	true	**true**	←
true	–	–	true	true	–	–	
true	–	true	true	true	–	–	
true	true	–	–	true	true	**true**	←
true	true	true	true	true	true	–	

Seq. plan!

Seq. plan!

Encoding State Sequences using State Propositions

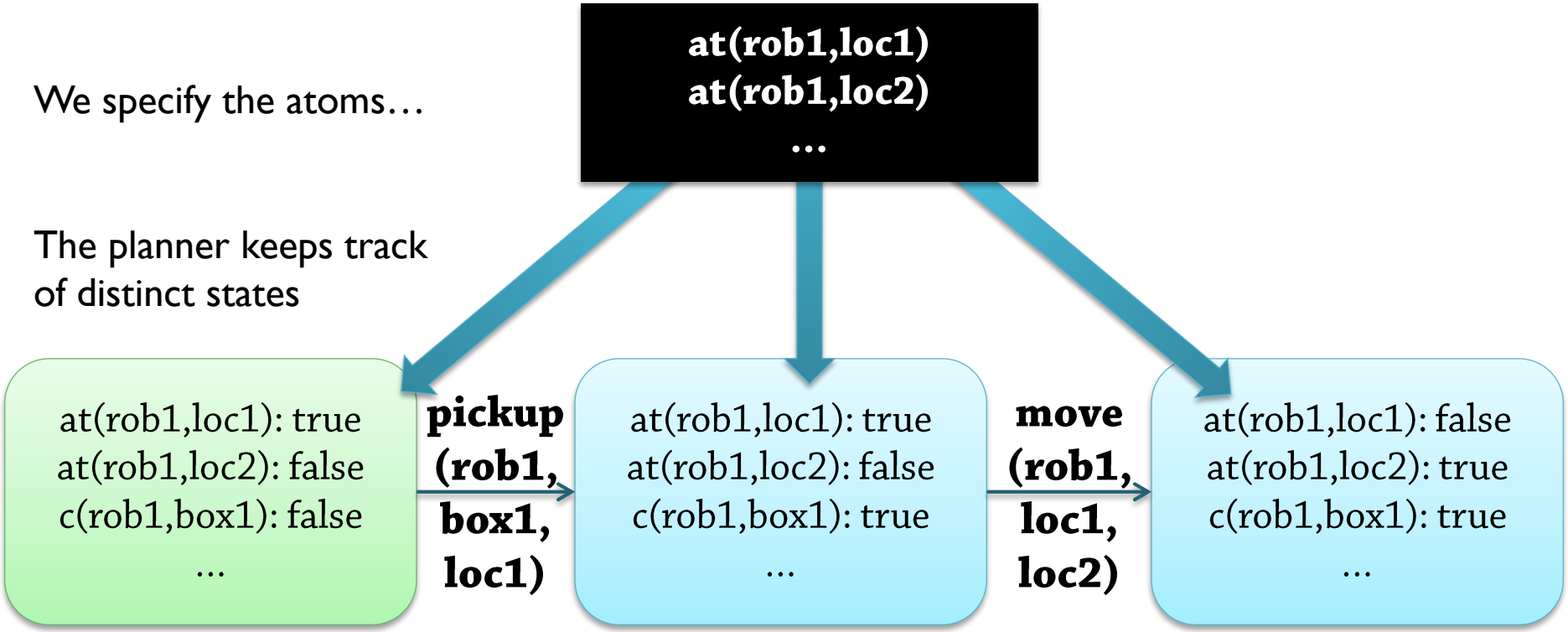
Multiple States

jonkv@ida

- **Planning** involves **multiple states**!
  - Ordinary planners handle this implicitly
    - We just say ”at(rob1,loc1)"
    - The planner keeps track of **which state** we mean
    - Example: Forward-chaining

We specify the atoms…

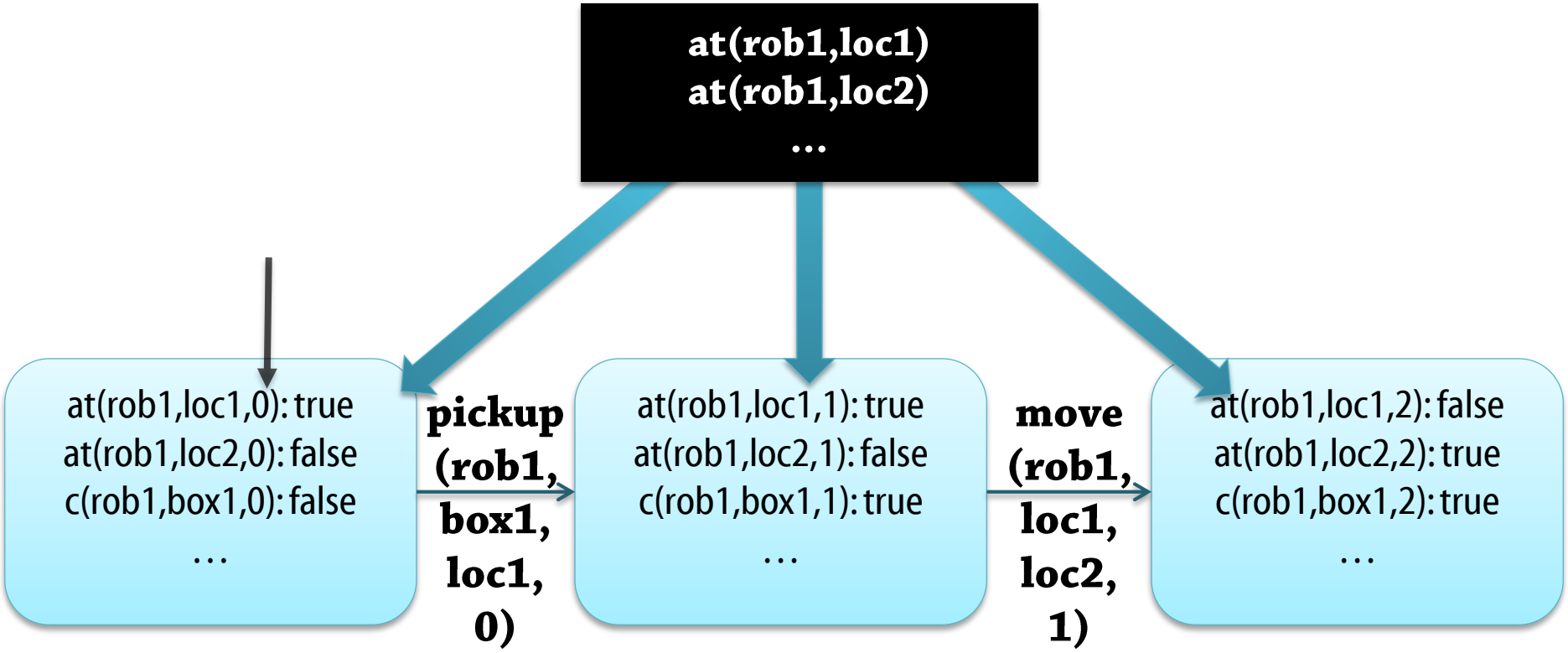
The planner keeps track of distinct states



Multiple States (2)

jonkv@ida

- SAT solvers have **no concept of separate states**!
  - Each **assignment** must correspond to an **entire state sequence**
  - In the translation,
    create one fact proposition for **each fact and state**
    and one action proposition for **each action and "plan step"**



Multiple States (3)

jonkv@ida

- Now: One sequence of states ⇔ one assignment

SAT assignment

at(rob1,loc1,0): true
at(rob1,loc2,0): false
c(rob1,box1,0): false
at(rob1,loc1,1): true
at(rob1,loc2,1): false
c(rob1,box1,1): true
at(rob1,loc1,2): false
at(rob1,loc2,2): true
c(rob1,box1,2): true
…
pickup(rob1,box1,loc1,0)
move(rob1,loc1,loc2,1)
…

at(rob1,loc1)
at(rob1,loc2)
…

Our example problem has 5 atoms

at(rob1,loc1)
at(rob1,loc2)
at(box1,loc1)
at(box1,loc2)
carrying(rob1, box1)

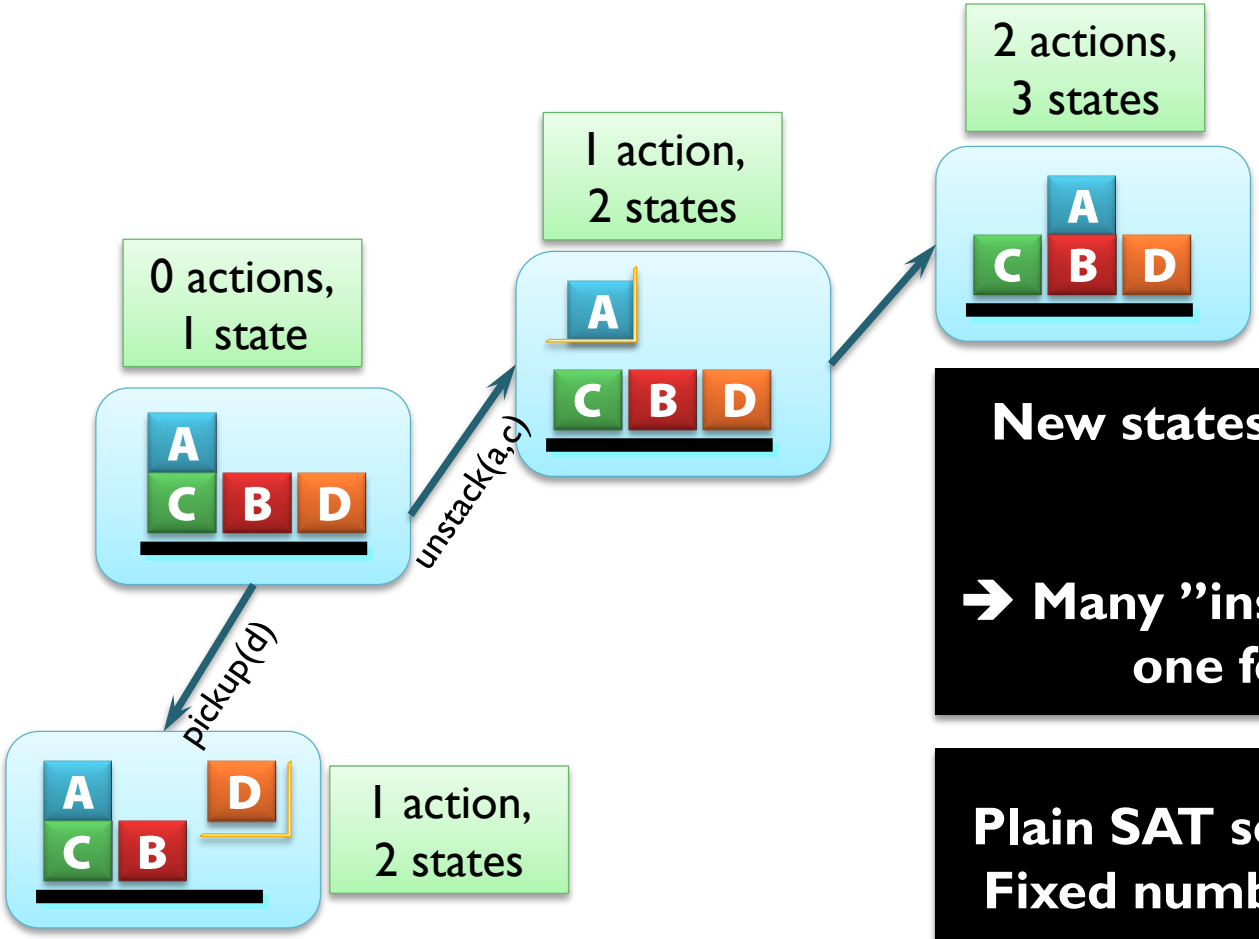
5 propositions for each timepoint
2^5=32 possible assignments for each timepoint
$32^{(n+1)}$ possible assignments in total,
where n = number of actions in the sequence

Bounded Planning

Solution Length

jonkv@ida

- We don't know in advance **how long** a solution will be!
  - Planners must handle action sequences of **varying length**
    - Forward-chaining example:



New states are "allocated" as needed

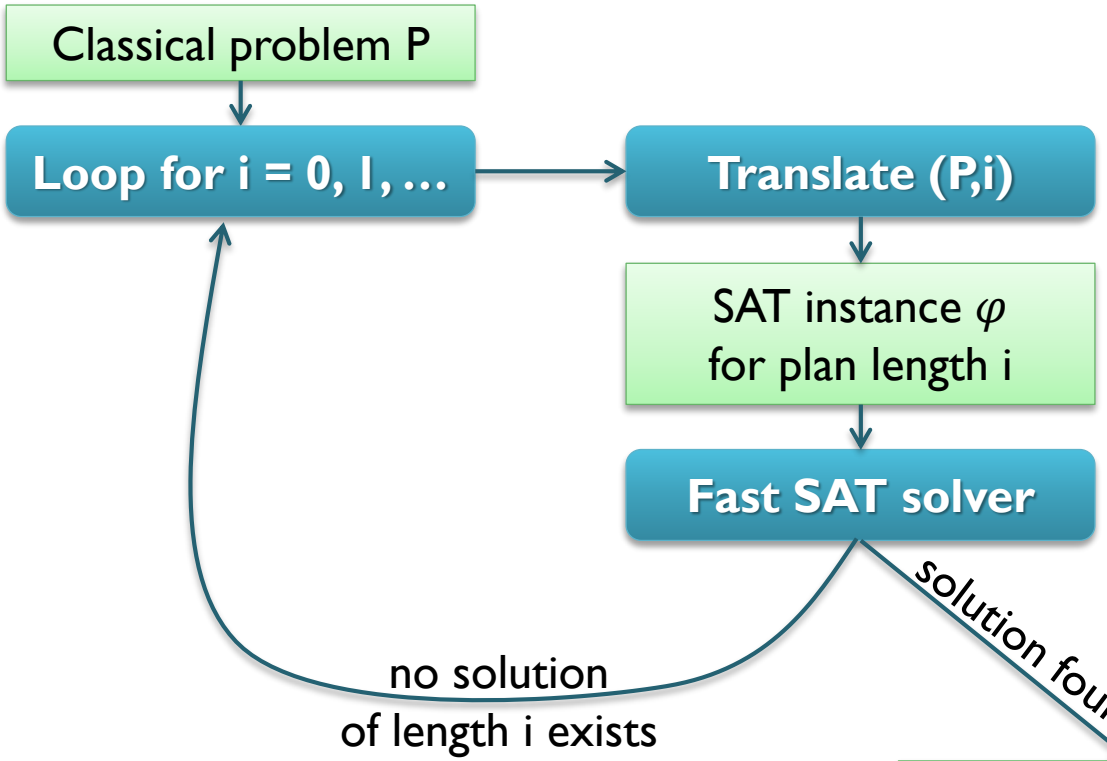
➔ Many "instances" of on(A,B), one for each state

Plain SAT solvers don't do this: Fixed number of propositions!

Iterative Search

jonkv@ida

- Again, use a form of **iterative deepening** search



if φ is satisfiable,
we get an assignment of truth values satisfying φ
➔ analyze it to see which actions should be part of the corresponding plan
➔ return the solution plan

{ solutions to P } =
{ solutions to $(P, i) \mid i \in \mathbb{N}$ }

Iterative Search

jonkv@ida

- Alternatives exist!
  - Test plan lengths 1, 2, 4, 8, 16, …
    - Works if we don't require an action in each step (*max* length, not *exact* length)
  - Test many plan lengths in parallel, one search process per CPU core

    Try to find plan of length 10

    Try to find plan of length 11

    Making use of all cores is comparatively simple
  - Test many plan lengths in parallel, *multiple* searches per CPU core

    Try to find plan of length 10

    None exists; takes a long time to verify

    Try to find plan of length 11

    Many exist; one is found quickly

    Can save time overall, even though each search is slower
  - …

Remaining Problem

jonkv@ida

- Remaining problem to solve:
  - Using propositional satisfiability solvers
    to find a **solution** with **exactly** n actions and $n + 1$ states

    1. Finding **executable action sequences** with exactly n actions

    2. Finding **solutions** among the executable action sequences

Finding Executable Action Sequences with Exactly n Actions

Representation Overview

jonkv@ida

- We have not added any formulas!
  - ➔ **Every** SAT *assignment* is a SAT *solution*…

Fact propo-sitions

at(rob1,loc1,0):	true/false
at(rob1,loc2,0):	true/false
at(box1,loc1,0):	true/false
at(box1,loc2,0):	true/false
carrying(rob1,box1,0):	true/false
…	
…	
at(rob1,loc1,*n*):	true/false
at(rob1,loc2,*n*):	true/false
at(box1,loc1,*n*):	true/false
at(box1,loc2,*n*):	true/false
carrying(rob1,box1,*n*):	true/false

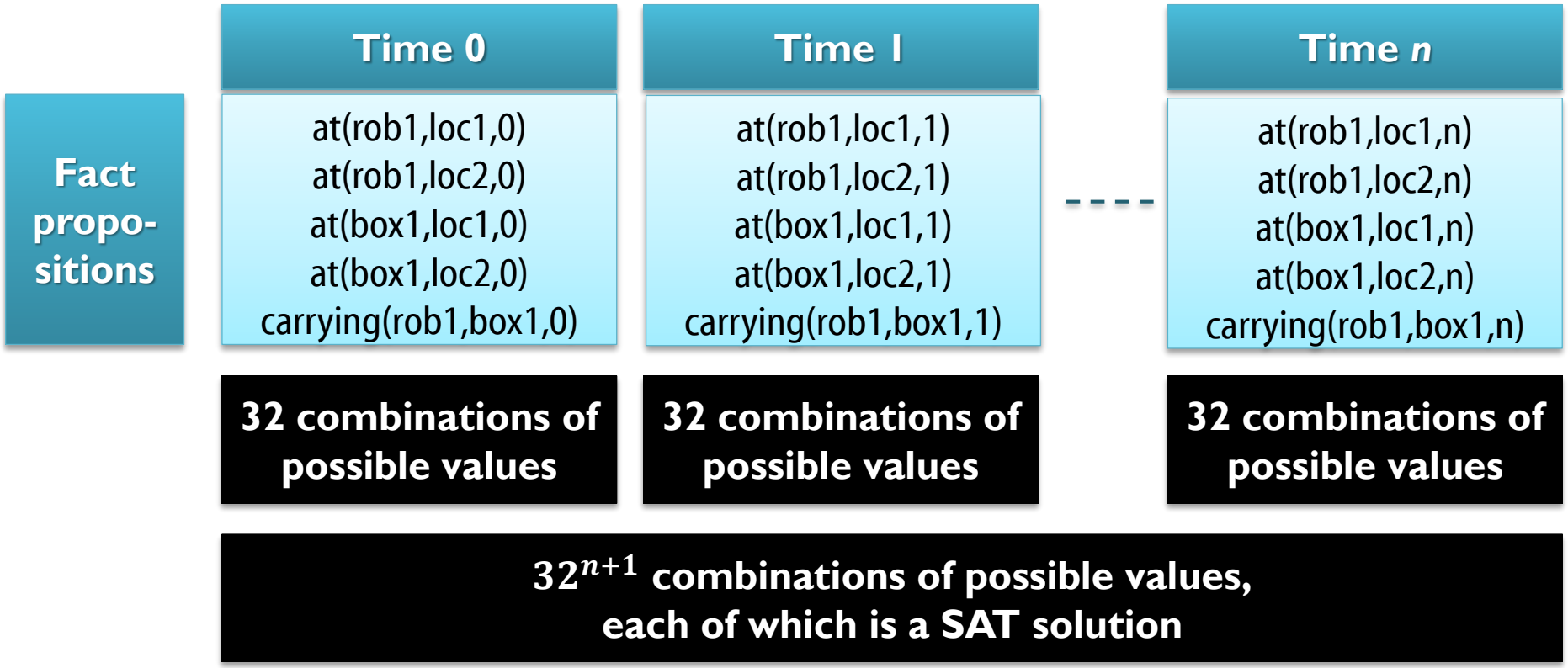
$$2^{5(n+1)} = 32^{n+1}$$

combinations of possible truth values, each of which is a SAT *solution*

Representation Overview (2)

jonkv@ida

- We can *visualize* an assignment as "time-based"
  - Even though the SAT solver only sees a single set of propositions…



Formulas in φ: Initial State

jonkv@ida

- We specify the **initial state**
  - Notation:
    - L = { **all** atoms in the planning problem instance }
    - s_0 = { atoms that are **true** in the initial state } (classical initial state)
  - For the example:
    - L = {at(rob1,loc1), at(rob1,loc2), at(box1,loc1), at(box1,loc2), carrying(rob1,box1)}
    - s_0 = {at(rob1,loc1), at(box1,loc2) }
    - Formula given to SAT solver:
      at(rob1,loc1,0) ∧ at(box1,loc2,0) ∧
      ¬at(rob1,loc2,0) ∧ ¬at(box1,loc1,0) ∧
      ¬carrying(rob1,box1,0)
  - General formula:
    - $\bigwedge\{atom_0 | atom \in s_0\} \wedge \bigwedge\{\neg atom_0 | atom \in L - s_0\}$

Propositions at time **zero**!

Negative facts must be included:
SAT solvers do not assume
what is ”missing” must be false

- If l is a literal, then l_i is the corresponding proposition for state s_i
- If l = at(rob1,loc1) then l_{12} = at(rob1,loc1,12)

Representation Overview

jonkv@ida

- Now **only** assignments satisfying the initial state formula are SAT solutions

	Time 0	Time 1	Time *n*
Fact propo-sitions	at(rob1,loc1,0) at(rob1,loc2,0) at(box1,loc1,0) at(box1,loc2,0) carrying(rob1,box1,0)	at(rob1,loc1,1) at(rob1,loc2,1) at(box1,loc1,1) at(box1,loc2,1) carrying(rob1,box1,1)	at(rob1,loc1,n) at(rob1,loc2,n) at(box1,loc1,n) at(box1,loc2,n) carrying(rob1,box1,n)
	Specified by the formula	32 combinations of possible values	32 combinations of possible values

32^n combinations of possible values, each of which is a SAT solution

Action Propositions

jonkv@ida

- Satisfiability has no concept of "**finding actions**"!
  - Additional propositions encode whether a **specific action** is executed at a **specific timepoint** or not
    - **move(rob1,loc2,loc1,0)** is true iff **move(rob1,loc2,loc1)** is executed at time 0
    - **move(rob1,loc2,loc1,1)** is true iff **move(rob1,loc2,loc1)** is executed at time 1
    - **move(rob1,loc2,loc1,2)** …
    - …
    - **move(rob1,loc2,loc1,*(n-1)*)**

No action proposition for *n*! Why?

  - The SAT solver will assign values to these propositions
    - Determines which actions are executed, and when

Representation Overview

jonkv@ida

	Time 0	Time 1	Time *n*
Fact propo-sitions	at(rob1,loc1,0) at(rob1,loc2,0) at(box1,loc1,0) at(box1,loc2,0) carrying(rob1,box1,0)	at(rob1,loc1,1) at(rob1,loc2,1) at(box1,loc1,1) at(box1,loc2,1) carrying(rob1,box1,1)	at(rob1,loc1,n) at(rob1,loc2,n) at(box1,loc1,n) at(box1,loc2,n) carrying(rob1,box1,n)
	Completely defined	32 combinations	32 combinations
Action propo-sitions	move(rob1,loc1,loc1,0) move(rob1,loc1,loc2,0) move(rob1,loc2,loc1,0) move(rob1,loc2,loc2,0) pickup(rob1,box1,l1,0) pickup(rob1,box1,l2,0)	move(rob1,loc1,loc1,1) move(rob1,loc1,loc2,1) move(rob1,loc2,loc1,1) move(rob1,loc2,loc2,1) pickup(rob1,box1,l1,1) pickup(rob1,box1,l2,1)	
	64 combinations	**64 combinations**	

Formulas in φ: Sequential Plans

jonkv@ida

- We are considering **sequential** planning
  - Ensured through a ***complete exclusion axiom***:
    - No **pair** of actions can be executed at any timepoint
    - ➔ For all actions a and b and for all timepoints $i < n$, we require $\neg a_i \vee \neg b_i$
  - For the example, with $n = 1$:
    - **¬move(rob1,loc1,loc2,0) ∨ ¬move(rob1,loc2,loc1,0)**
    - …

Complete exclusion: All parallelism forbidden

Alternative: *Conflict exclusion axioms*
may only prevent actions from being executed in parallel if they have conflicts

Representation Overview

jonkv@ida

	Time 0	Time 1		Time *n*
Fact propo-sitions	at(rob1,loc1,0) at(rob1,loc2,0) at(box1,loc1,0) at(box1,loc2,0) carrying(rob1,box1,0)	at(rob1,loc1,1) at(rob1,loc2,1) at(box1,loc1,1) at(box1,loc2,1) carrying(rob1,box1,1)	- - - -	at(rob1,loc1,n) at(rob1,loc2,n) at(box1,loc1,n) at(box1,loc2,n) carrying(rob1,box1,n)
	Completely defined	32 combinations		32 combinations
Action propo-sitions	move(rob1,loc1,loc1,0) move(rob1,loc1,loc2,0) move(rob1,loc2,loc1,0) move(rob1,loc2,loc2,0) pickup(rob1,box1,l1,0) pickup(rob1,box1,l2,0)	move(rob1,loc1,loc1,1) move(rob1,loc1,loc2,1) move(rob1,loc2,loc1,1) move(rob1,loc2,loc2,1) pickup(rob1,box1,l1,1) pickup(rob1,box1,l2,1)	- - - -	
	7 alternatives	**7 alternatives**		

Now we need formulas to **relate** these propositions to each other!

Formulas in φ: Action Preconditions

jonkv@ida

- For **every action** *a* and **every timepoint** *i* < *n*:
  - **If** the precondition of *a* is false in state *i*,
    then *a* cannot be executed at step *i*
    - precond(a) false in state *i* ➔ *a* not executed in step *i*
    - Multiple formulas: $\{ \neg p_i \rightarrow \neg a_i \mid p \quad \text{precond}(a) \}$
    - Example: { ¬at(from, 0) → ¬drive(from, to, 0),
      ¬have-fuel(0) → ¬drive(from, to, 0) }
  - Logically equivalent:
    if a executed in step *i* **then** $\text{precond}(a)$ true in state *i*
    - Single formula: $a_i \rightarrow \bigwedge \{ p_i \mid p \quad \text{precond}(a) \}$
    - Example: drive(from, to, 0) → at(from, 0) ∧ have-fuel(0)
  - There are **SAT assignments** where:
    - precond(a) is false in state i
    - a is executed in step i
  - But these assignments do not satisfy all formulas
    ➔ are not **solutions**

Formulas in φ: Action Effects

jonkv@ida

- For **every action** a and **every timepoint** $i < n$:
  - **If** a is executed at step i,
    then the **effects** of a must be true in state $i + 1$
  - Formula:
    - $a_i \rightarrow \bigwedge \{ e_{i+1} | \; e \quad \text{effects}(a) \}$

Formulas in φ: Action Example

jonkv@ida

$$a_i \rightarrow \bigwedge \{pi \mid p \quad \text{precond}(a)\} \wedge \bigwedge \{e_{i+1} \mid e \quad \text{effects}(a)\}$$

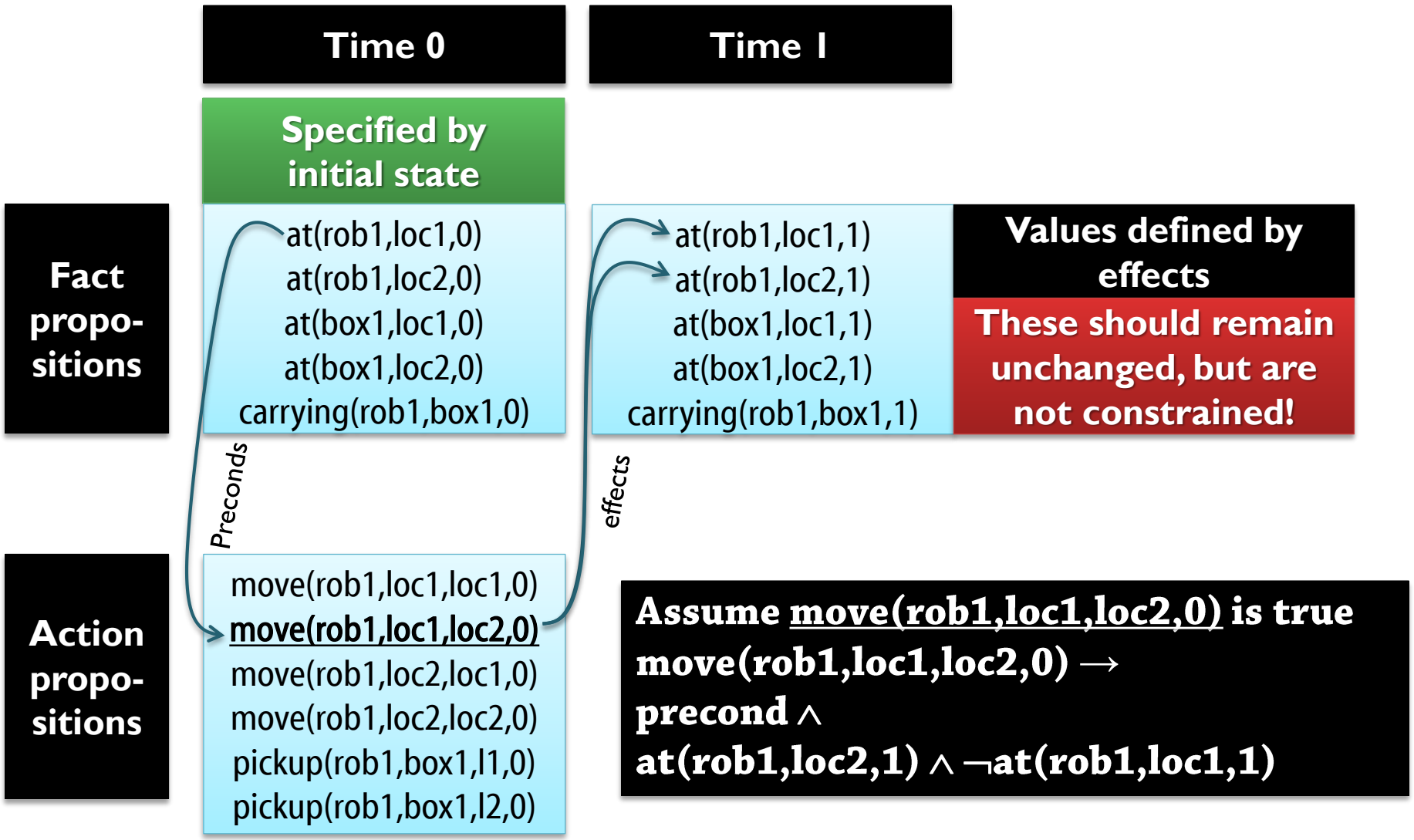
- For the **move** action, with $n = 2$ (plans of length 2):

	action	precond	effects	
	move(rob1,loc1,loc2,0)	at(rob1,loc1,0)	at(rob1,loc2,1) ∧ ¬at(rob1,loc1,1)	time 0—1
	move(rob1,loc2,loc1,0)	at(rob1,loc2,0)	at(rob1,loc1,1) ∧ ¬at(rob1,loc2,1)	
***	move(rob1,loc1,loc1,0)	at(rob1,loc1,0)	at(rob1,loc1,1) ∧ ¬at(rob1,loc1,1)	
	move(rob1,loc2,loc2,0)	at(rob1,loc2,0)	at(rob1,loc2,1) ∧ ¬at(rob1,loc2,1)	
	move(rob1,loc1,loc2,1)	at(rob1,loc1,1)	at(rob1,loc2,2) ∧ ¬at(rob1,loc1,2)	time 1—2
	move(rob1,loc2,loc1,1)	at(rob1,loc2,1)	at(rob1,loc1,2) ∧ ¬at(rob1,loc2,2)	
***	move(rob1,loc1,loc1,1)	at(rob1,loc1,1)	at(rob1,loc1,2) ∧ ¬at(rob1,loc1,2)	
	move(rob1,loc2,loc2,1)	at(rob1,loc2,1)	at(rob1,loc2,2) ∧ ¬at(rob1,loc2,2)	

- Formulas marked with "***" have *inconsistent consequences*
  - Formula 3 equivalent to ¬move(rob1,loc1,loc1,0), etc.

Representation Overview: Closer Look

jonkv@ida



Unchanged Facts

jonkv@ida

- Again: The SAT solver has no notion of states or "unchanged"
  - We must explicitly **say** that unaffected propositions remain the same
  - We need *frame axioms*

- For example, **explanatory** frame axioms

If there is a **change**…	…there must be an **explanation**.

  - $\neg at(rob1,loc1,0) \wedge at(rob1,loc1,1) \rightarrow move(rob1,loc2,loc1,0)$
    $\neg at(rob1,loc2,0) \wedge at(rob1,loc2,1) \rightarrow move(rob1,loc1,loc2,0)$
    $at(rob1,loc1,0) \wedge \neg at(rob1,loc1,1) \rightarrow move(rob1,loc1,loc2,0)$
    $at(rob1,loc2,0) \wedge \neg at(rob1,loc2,1) \rightarrow move(rob1,loc2,loc1,0)$

  - If rob1 isn't at loc1 at time 0, but it **is** at loc1 at time 1,
    then there must be an **explanation**:
    We executed move(rob1,loc2,loc1) at time 0!

Frame Axioms

jonkv@ida

- **Explanatory frame axioms:**
  - One formula for every atom l and every timepoint $i < n$
  - If l **changes** between s_i and s_{i+1}, then the action at step i must be responsible:

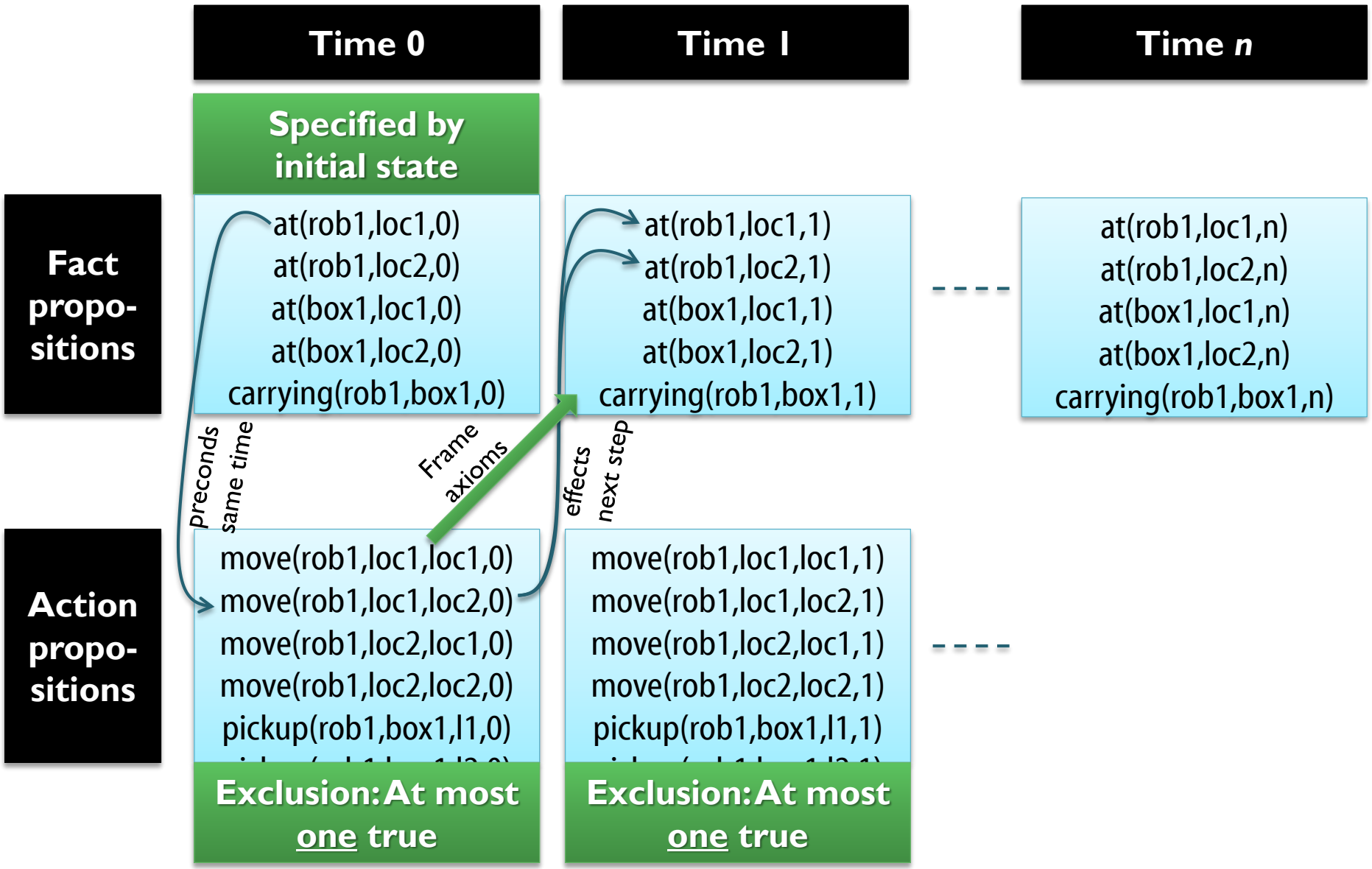
$$(\neg l_i \wedge l_{i+1} \rightarrow \vee\{\, a_i \mid a \in A \wedge l \quad \text{effects}^+(a)\,\})$$
$$\wedge\, (l_i \wedge \neg l_{i+1} \rightarrow \vee\{\, a_i \mid a \in A \wedge l \quad \text{effects}^-(a)\,\})$$

In general there may be more than one possible cause ➔
a **disjunction** to the right of →

Example:
¬at(me,loc1,3) ∧ at(me,loc1,4) => walk(3) ∨ run(3) ∨ drive(3)

Representation Overview

jonkv@ida



Finding Solutions of Fixed Length

Finding Solutions

jonkv@ida

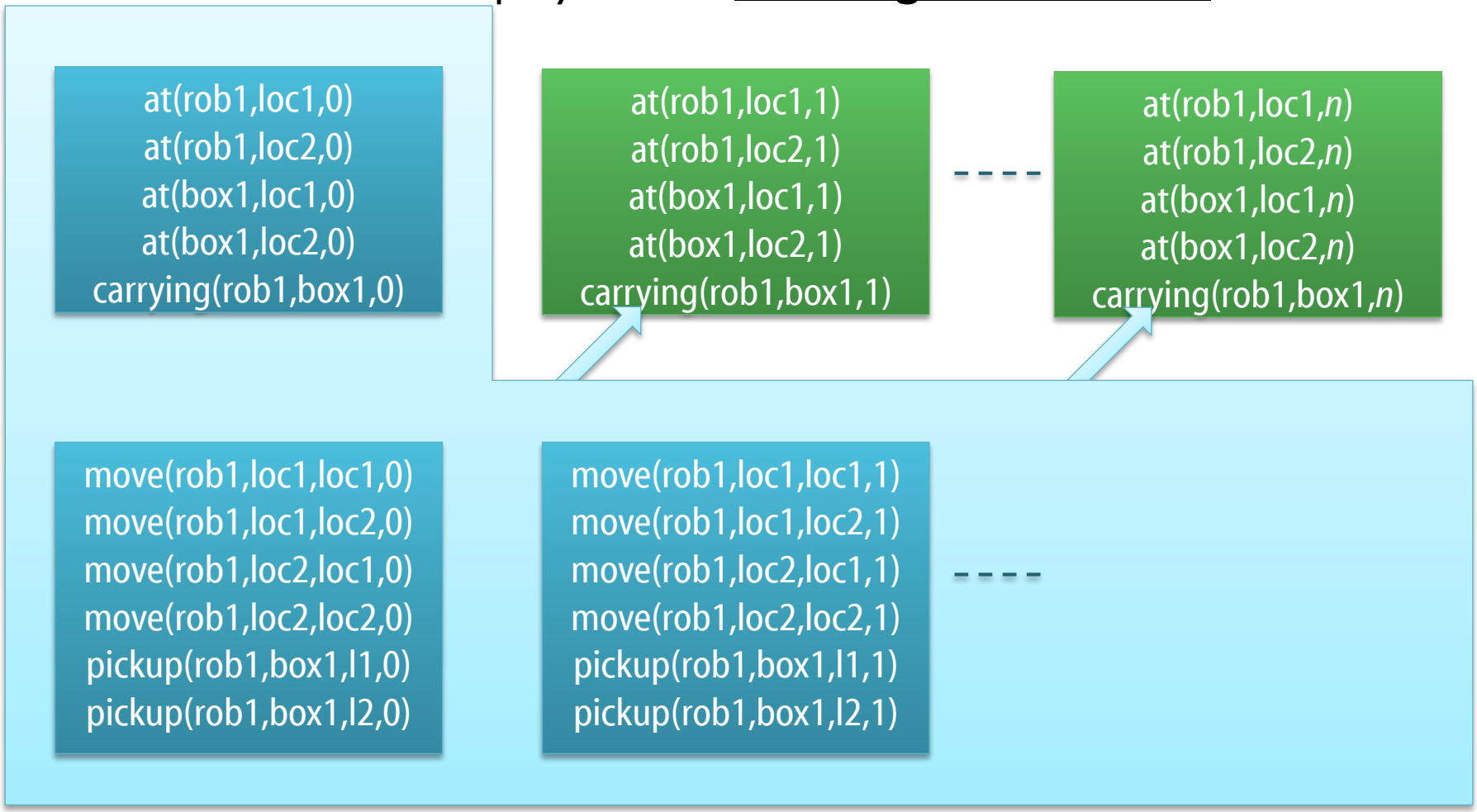
- With the current encoding for the problem (P, n):
  - We have one **SAT solution**
    for every **executable action sequence** of length n

- But we want:
  - One **SAT solution**
    for every **solution** of length n

Completely Defined States

jonkv@ida

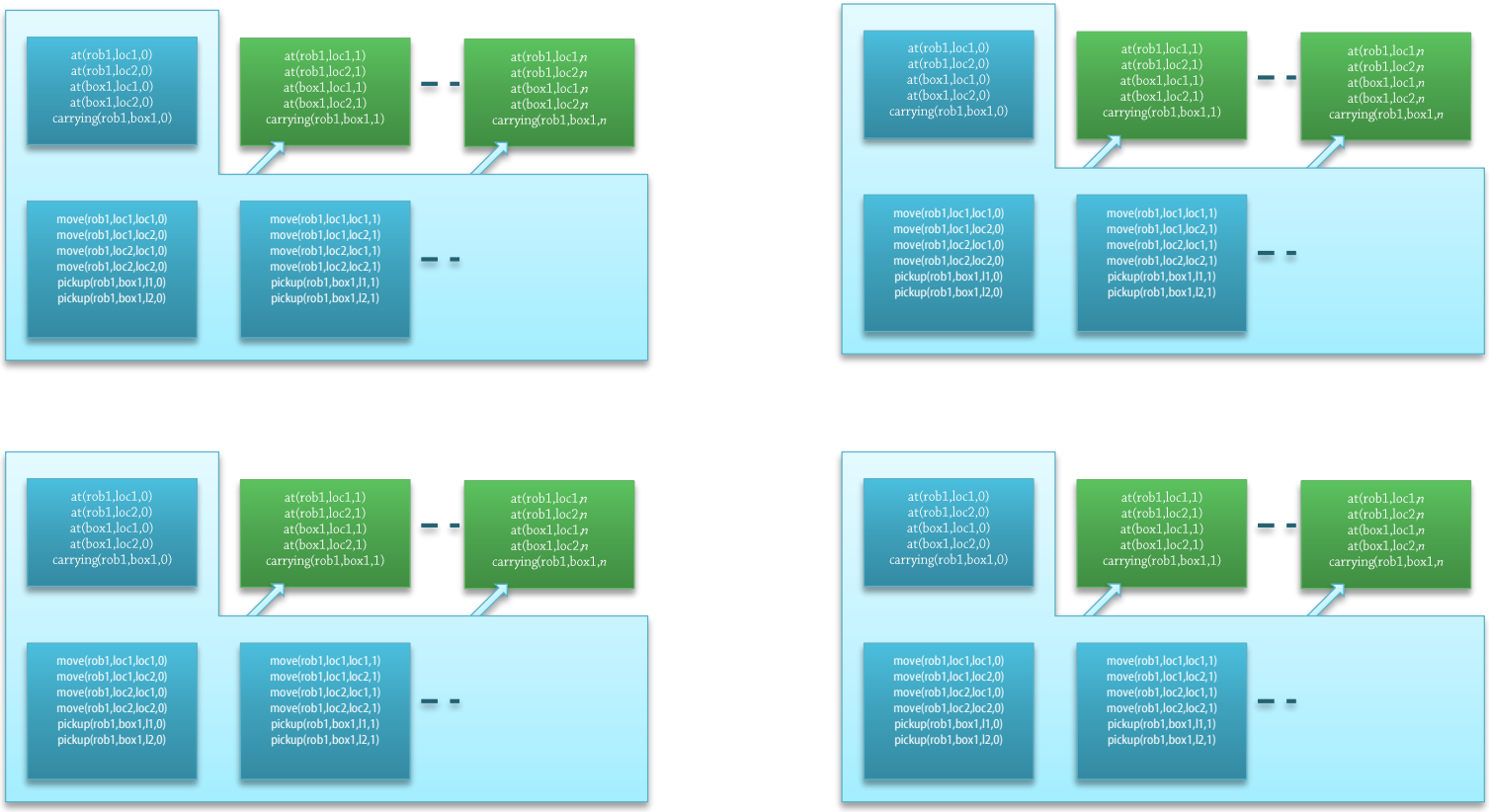
- We are doing **deterministic** planning:
  - Given an **initial state** and an **assignment to action propositions**, **all** other states are uniquely defined, **including the final state**



Executable Action Sequences

jonkv@ida

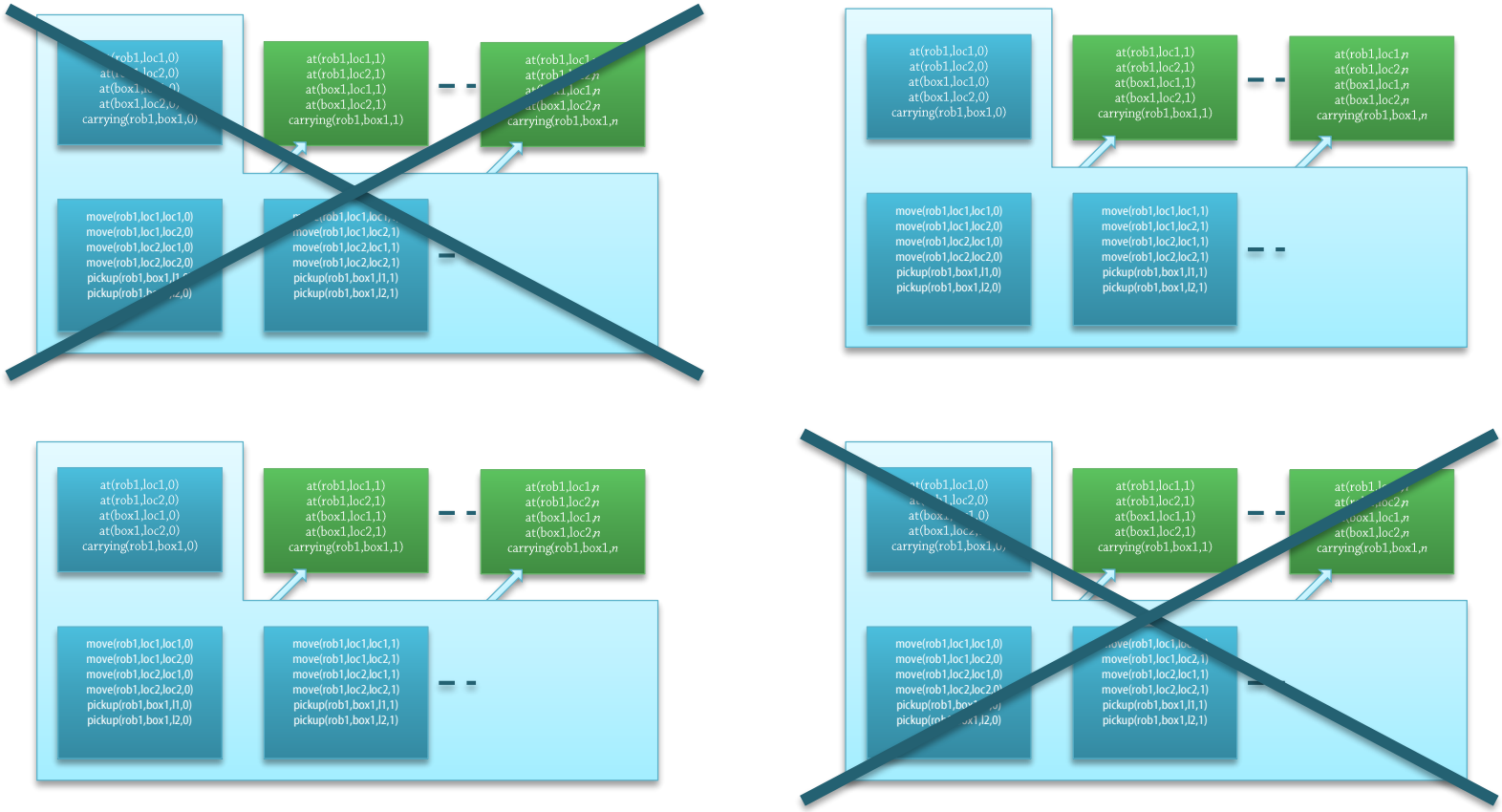
- Suppose you have 4 SAT solutions for the *current* formulas
  - Each one **must** correspond to a **different** executable action sequence



Solution Plans

jonkv@ida

- If we **remove** those where the last state does not satisfy the goal
  - All of the **remaining** ones correspond to **solution plans**, *guaranteed* to achieve the goal



For this to work, we must have determinism: One SAT solution per plan!

Formulas in φ: Goal

jonkv@ida

- To remove **SAT solutions** that don't satisfy the goal:
  - State that the goal formula must be true
    - $\wedge\{lit_n \mid lit \in g^+\} \wedge$
      $\wedge\{\neg lit_n \mid lit \in g^-\}$,
      where n is intended length of the plan (must hold at the end!)

- For the example:
  - If we are searching for plans of length 1:
    Goal: {carrying(rob1,box1)}
    Encoding: carrying(rob1,box1,1)
  - If we are searching for plans of length 5:
    Goal: {carrying(rob1,box1)}
    Encoding: carrying(rob1,box1,5)

Representation Overview

jonkv@ida

	Time 0	Time 1		Time *n*
	Specified by initial state			**Constrained (partly) by goal**
Fact propo-sitions	at(rob1,loc1,0) at(rob1,loc2,0) at(box1,loc1,0) at(box1,loc2,0) carrying(rob1,box1,0)	at(rob1,loc1,1) at(rob1,loc2,1) at(box1,loc1,1) at(box1,loc2,1) carrying(rob1,box1,1)	- - - -	at(rob1,loc1,n) at(rob1,loc2,n) at(box1,loc1,n) at(box1,loc2,n) carrying(rob1,box1,n)
Action propo-sitions	move(rob1,loc1,loc1,0) move(rob1,loc1,loc2,0) move(rob1,loc2,loc1,0) move(rob1,loc2,loc2,0) pickup(rob1,box1,l1,0) pickup(rob1,box1,l2,0)	move(rob1,loc1,loc1,1) move(rob1,loc1,loc2,1) move(rob1,loc2,loc1,1) move(rob1,loc2,loc2,1) pickup(rob1,box1,l1,1) pickup(rob1,box1,l2,1)	- - - -	

With Nondeterminism

jonkv@ida

- **If** we had allowed nondeterministic actions, incomplete states
  - **One** action sequence could yield **many SAT solutions** representing **different outcomes**
    - Initially: heads
    - Your goal: tails

SAT solution 1

Time 0	Time 1
heads	tails
toss-coin(0)	

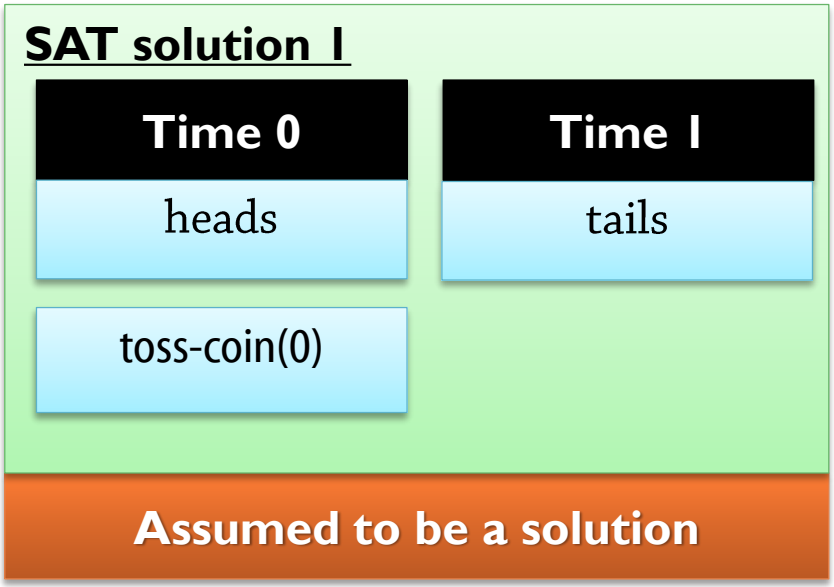
SAT solution 2

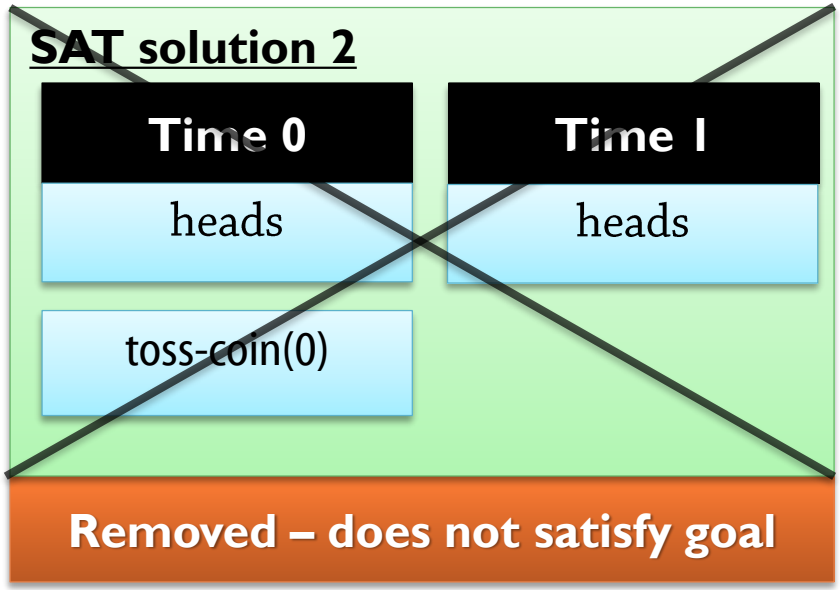
Time 0	Time 1
heads	heads
toss-coin(0)	

With Nondeterminism (2)

jonkv@ida

- **If** we had allowed nondeterministic actions, incomplete states
  - Adding the goal formula would **filter out undesirable outcomes**
  - Would find plans that *may* achieve the goal



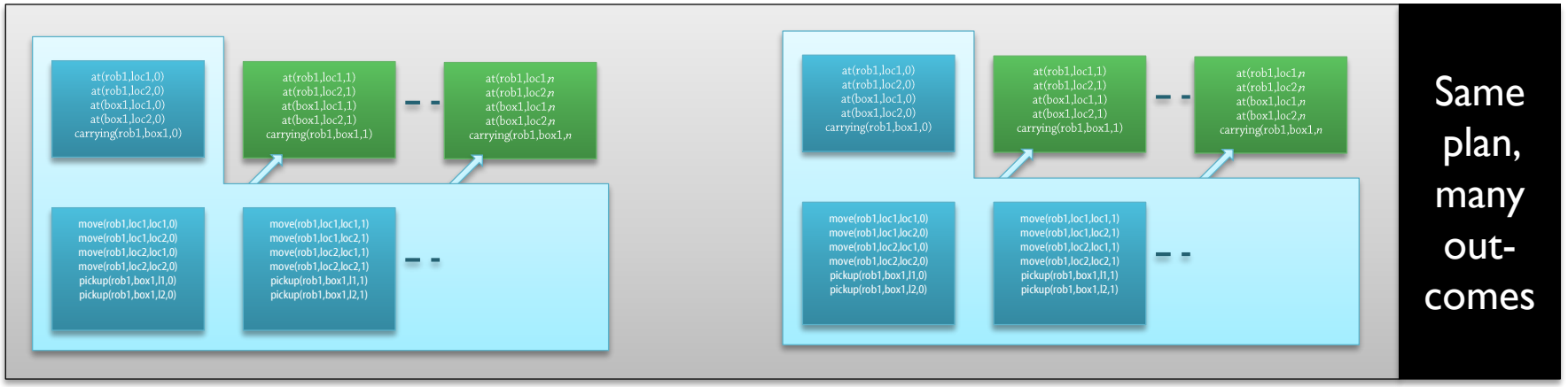


Given determinism,
only "bad" action sequences are filtered out –
not undesirable outcomes

With Nondeterminism (3)

jonkv@ida

- One (very inefficient) approach to nondeterminism:
  - Generate **all** solutions
  - **Group** them by plan
  - Check if **all** outcomes in a group satisfy the goal



Example

Creating a Single-Step Plan: Problem

jonkv@ida

Formulas in the SAT problem

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

Goal

carrying(rob1,box1,1)

Complete exclusion

¬move(rob1,loc1,loc2,0)
∨
¬move(rob1,loc2,loc1,0),
…, …, …

Frame Axioms

(¬carrying(rob1,box1,0) ∧ carrying(rob1,box1,1) →
pickup(rob1,box1,l1,0) ∨ pickup(rob1,box1,l2,0)) ∧ …

Propositions in a SAT assignment (under construction!)

at(rob1,loc1,0)	?
at(rob1,loc2,0)	?
at(box1,loc1,0)	?
at(box1,loc2,0)	?
carrying(rob1,box1,0)	?

move(rob1,loc1,loc1,0)	?
move(rob1,loc1,loc2,0)	?
move(rob1,loc2,loc1,0)	?
move(rob1,loc2,loc2,0)	?
pickup(rob1,box1,l1,0)	?
pickup(rob1,box1,l2,0)	?

at(rob1,loc1,1)	?
at(rob1,loc2,1)	?
at(box1,loc1,1)	?
at(box1,loc2,1)	?
carrying(rob1,box1,1)	?

1-Step Plan: First Propagations

jonkv@ida

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Necessary consequences

at(rob1,loc1,0)	true
at(rob1,loc2,0)	false
at(box1,loc1,0)	true
at(box1,loc2,0)	false
carrying(rob1,box1,0)	false

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

move(rob1,loc1,loc1,0)	false
move(rob1,loc1,loc2,0)	true
move(rob1,loc2,loc1,0)	
move(rob1,loc2,loc2,0)	
pickup(rob1,box1,l1,0)	
pickup(rob1,box1,l2,0)	

Goal

carrying(rob1,box1,1)

Necessary consequences

at(rob1,loc1,1)	
at(rob1,loc2,1)	
at(box1,loc1,1)	
at(box1,loc2,1)	
carrying(rob1,box1,1)	true

Try move(rob1,loc1,loc1,0)=true ➔ contradiction in effects

Try move(rob1,loc1,loc2,0)=true ➔ seems OK so far

Choices made in SAT search – these are backtrack points!

1-Step Plan: Action ➔ Preconds, Effects

jonkv@ida

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

Goal

carrying(rob1,box1,1)

at(rob1,loc1,0)	true
at(rob1,loc2,0)	false
at(box1,loc1,0)	true
at(box1,loc2,0)	false
carrying(rob1,box1,0)	false

move(rob1,loc1,loc1,0)	false
move(rob1,loc1,loc2,0)	true
move(rob1,loc2,loc1,0)	
move(rob1,loc2,loc2,0)	
pickup(rob1,box1,l1,0)	
pickup(rob1,box1,l2,0)	

at(rob1,loc1,1)	false
at(rob1,loc2,1)	true
at(box1,loc1,1)	
at(box1,loc2,1)	
carrying(rob1,box1,1)	true

Complete exclusion

¬move(rob1,loc1,loc2,0) ∨
¬move(rob1,loc2,loc1,0),
…, …, …

Frame Axioms

(¬carrying(rob1,box1,0) ∧ carrying(rob1,box1,1) →
pickup(rob1,box1,l1,0) ∨ pickup(rob1,box1,l2,0)) ∧ …

1-Step Plan: Action ➔ Complete Exclusion

jonkv@ida

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

Goal

carrying(rob1,box1,1)

at(rob1,loc1,0)	true
at(rob1,loc2,0)	false
at(box1,loc1,0)	true
at(box1,loc2,0)	false
carrying(rob1,box1,0)	false

move(rob1,loc1,loc1,0)	false
move(rob1,loc1,loc2,0)	true
move(rob1,loc2,loc1,0)	false
move(rob1,loc2,loc2,0)	false
pickup(rob1,box1,l1,0)	false
pickup(rob1,box1,l2,0)	false

at(rob1,loc1,1)	false
at(rob1,loc2,1)	true
at(box1,loc1,1)	true
at(box1,loc2,1)	false
carrying(rob1,box1,1)	true

Complete exclusion

¬move(rob1,loc1,loc2,0) ∨
¬move(rob1,loc2,loc1,0),
…, …, …

Frame Axioms

(¬carrying(rob1,box1,0) ∧ carrying(rob1,box1,1) →
pickup(rob1,box1,l1,0) ∨ pickup(rob1,box1,l2,0)) ∧ …

1-Step Plan: Frame Axioms (1)

jonkv@ida

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

Goal

carrying(rob1,box1,1)

at(rob1,loc1,0)	true
at(rob1,loc2,0)	false
at(box1,loc1,0)	true
at(box1,loc2,0)	false
carrying(rob1,box1,0)	false

move(rob1,loc1,loc1,0)	false
move(rob1,loc1,loc2,0)	true
move(rob1,loc2,loc1,0)	false
move(rob1,loc2,loc2,0)	false
pickup(rob1,box1,l1,0)	false
pickup(rob1,box1,l2,0)	false

at(rob1,loc1,1)	false
at(rob1,loc2,1)	true
at(box1,loc1,1)	true
at(box1,loc2,1)	false
carrying(rob1,box1,1)	true

Complete exclusion

¬move(rob1,loc1,loc2,0) ∨
¬move(rob1,loc2,loc1,0),
…, …, …

Frame Axioms

(¬carrying(rob1,box1,0) ∧ carrying(rob1,box1,1) →
pickup(rob1,box1,l1,0) ∨ pickup(rob1,box1,l2,0)) ∧ …

1-Step Plan: Frame Axioms (2)

jonkv@ida

Initial state

at(rob1,loc1,0) ∧
¬at(rob1,loc2,0) ∧
¬carrying(rob1,box1,0) ∧
…

Action axioms

move(rob1,loc1,loc2,0) →
at(rob1,loc1,0) ∧
at(rob1,loc2,1) ∧
¬at(rob1,loc1,1),
…, …, …

Goal

carrying(rob1,box1,1)

at(rob1,loc1,0)	true
at(rob1,loc2,0)	false
at(box1,loc1,0)	true
at(box1,loc2,0)	false
carrying(rob1,box1,0)	false

move(rob1,loc1,loc1,0)	false
move(rob1,loc1,loc2,0)	true
move(rob1,loc2,loc1,0)	false
move(rob1,loc2,loc2,0)	false
pickup(rob1,box1,l1,0)	false
pickup(rob1,box1,l2,0)	false

at(rob1,loc1,1)	false
at(rob1,loc2,1)	true
at(box1,loc1,1)	true
at(box1,loc2,1)	false
carrying(rob1,box1,1)	true

Inconsistent!

No explanation
for changing carrying()

Complete exclusion

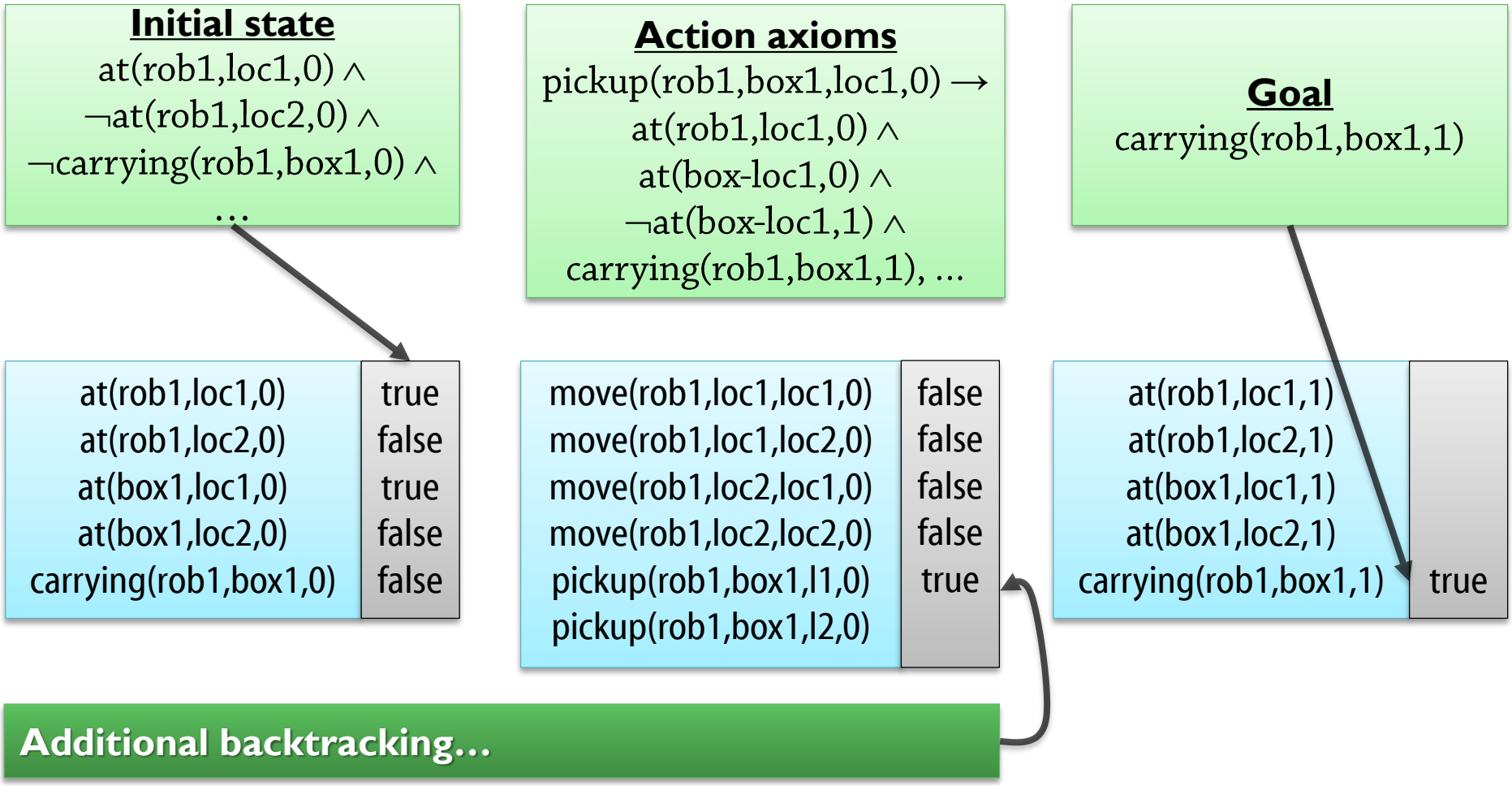
¬move(rob1,loc1,loc2,0) ∨
¬move(rob1,loc2,loc1,0),
…, …, …

Frame Axioms

(¬carrying(rob1,box1,0) ∧ carrying(rob1,box1,1) →
pickup(rob1,box1,l1,0) ∨ pickup(rob1,box1,l2,0)) ∧ …

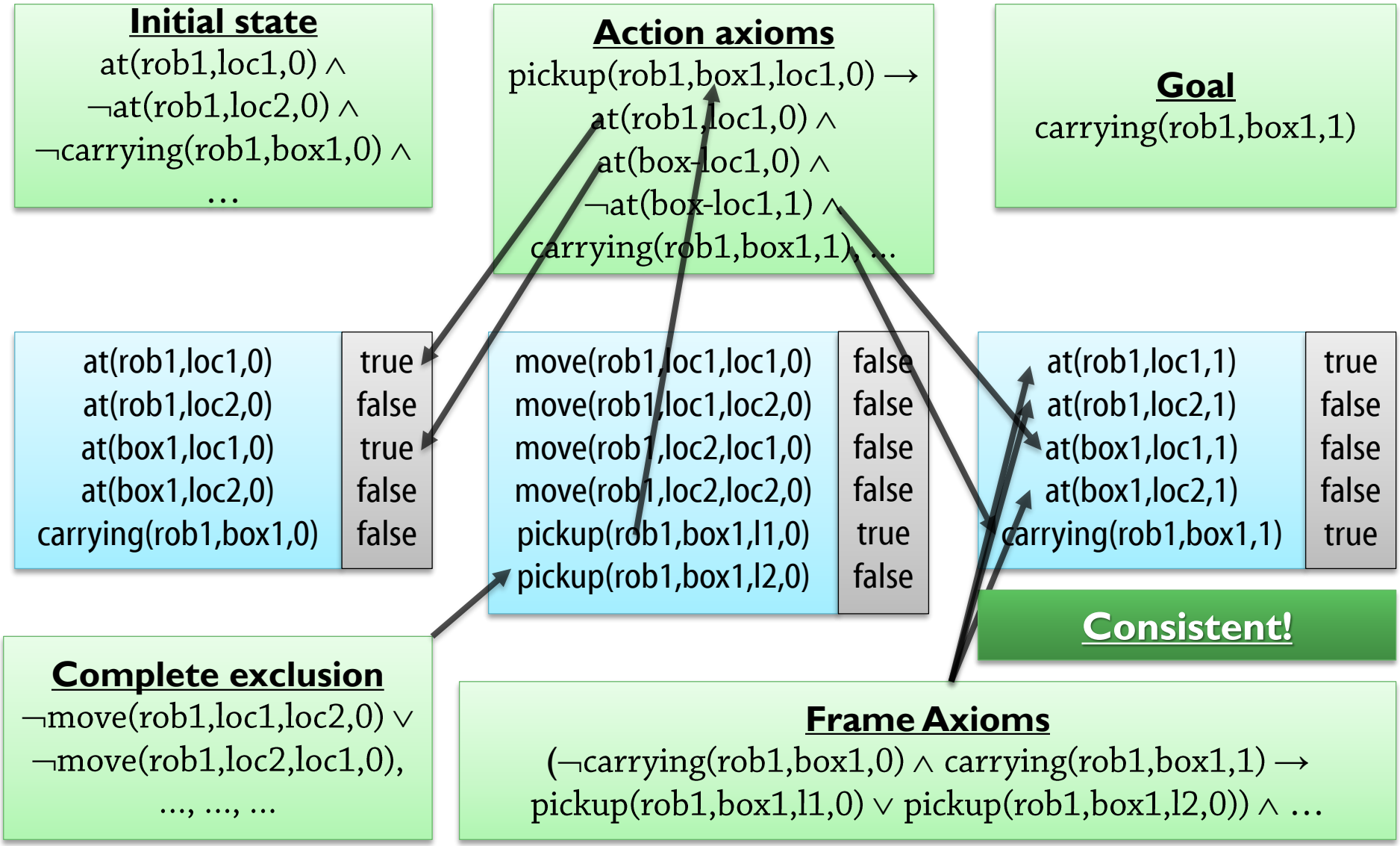
Creating a Single-Step Plan (3)

jonkv@ida



Creating a Single-Step Plan (4)

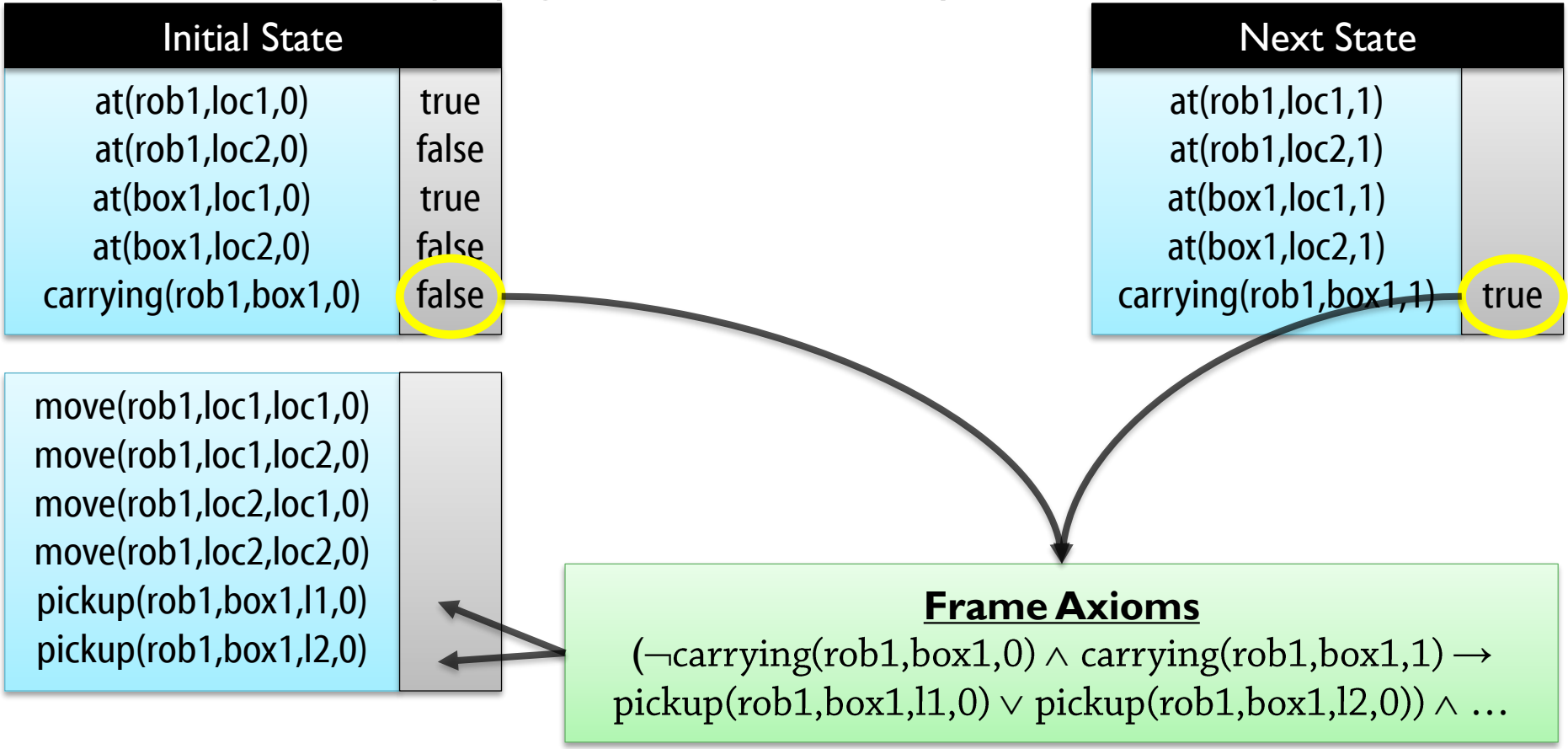
jonkv@ida



Advantages?

jonkv@ida

- What's the advantage?
  - SAT solvers can have very sophisticated search strategies
  - SAT solvers can propagate constraints "in any direction"

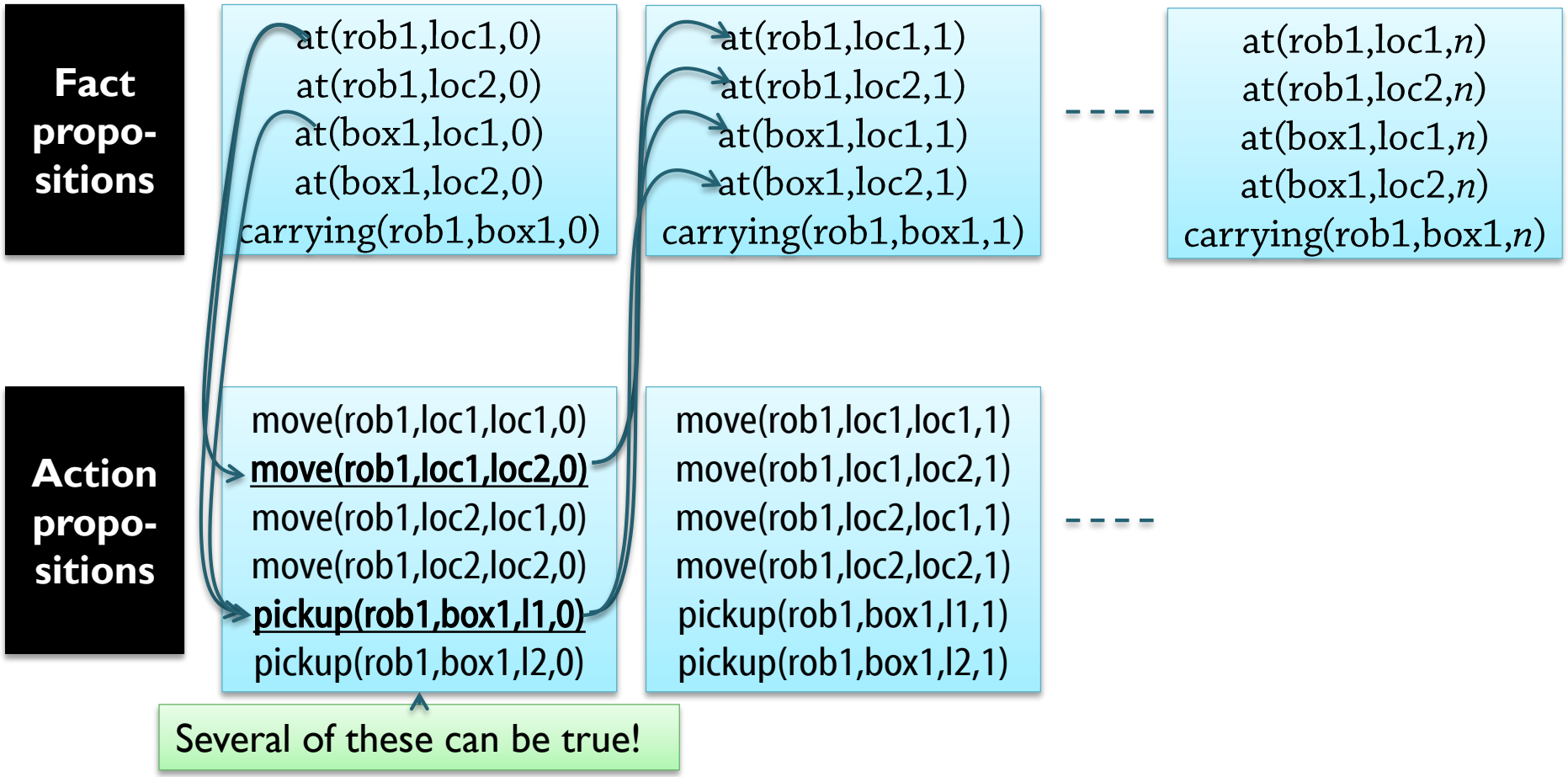


Concurrent Planning?

Concurrent Plans: Action Propositions

jonkv@ida

- SAT planning can be used to generate **concurrent** plans
  - Make many action propositions true at the same time step



Concurrent Plans: Constraints?

jonkv@ida

- Be very careful about semantics + constraints on concurrency!
  - If both move(rob1,loc1,*loc2*,0) and move(rob1,loc1,*loc3*,0) are true,
    then **both** at(rob1,loc1,0) ∧ at(rob1,*loc2*,1) ∧ ¬at(rob1,loc1,1)
    and at(rob1,loc1,0) ∧ at(rob1,*loc3*,1) ∧ ¬at(rob1,loc1,1) **must be true**
  - Equivalent to at(rob1,loc1,0) ∧ at(rob1,*loc2*,1) ∧ at(rob1,*loc3*,1) ∧ ¬at(rob1,loc1,1)
  - **Logically consistent**
    but results in a plan where we are at two places at the same time

Conflict Exclusion

jonkv@ida

- We must tell the SAT solver that this is not intended!
  - Requires *conflict exclusion axioms*
  - **∀-step plans**
    - Can execute a set of actions in a single time step if they are *pairwise independent*
    - ∀: Every execution order must work
    - Used in *GraphPlan*
  - **∃-step plans**
    - Can execute a set of actions in a single time step if there is *at least one* ordering where no action disables a later action or changes its conditional effects
    - Allows additional actions in the same "step"
    - A bit further from "execution parallelism" than ∀-step plans, but then, GraphPlan parallelism is not a very useful form of concurrency
  - Details not covered in this course!

Propositional Encodings

Propositionalization

jonkv@ida

- Technical issue: SAT solvers generally require **propositional** input
  - Not **first-order**: No variables, no parameters, no objects

- In planning, each type has a **finite and known** set of values
  - ➔ Each predicate has a finite and known set of instances
  - ➔ Can define a simple mapping
  - ➔ All **parameters** and **variables** disappear!

- **Convert** all first-order atoms to **propositions**
  - A first-order atom: at(rob1,loc1,4)
  - Becomes a proposition: at-rob1-loc1-4
- Including "action atoms"
  - A first-order "action atom": move(robot r, loc l, loc l', 4)
  - Becomes many atoms: move-rob1-loc1-loc1-4, move-rob1-loc1-loc2-4, …

Similar to the set-theoretic classical representation

Propositionalization (2)

jonkv@ida

- Looks almost identical – is there really any difference?

action:	**move(rob1, loc1, loc2, 3)**
effects:	**not at(rob1, loc1, 3),**
	at(rob1, loc2, 3)

action:	**move-rob1-loc1-loc2-3**
effects:	**not at-rob1-loc1-3,**
	at-rob1-loc2-3

- Remember that *names themselves are meaningless*
  - What happens if we replace every name?

action:	**a1(o1, o2, o3, o4)**
effects:	**not p1(o1, o2),**
	p1(o1, o3)

The planner still "sees" connections:
predicate **p1** affected,
object **o1** used several times, …

action:	**a2**
effects:	**not p2,**
	p3

Identifiers have no structure ➔
connections have disappeared!

What if we create a propositional planner or SAT solver that understands "-", picks out parts of identifiers?

That would simply be a *first-order* planner with a strange syntax…

Improvements

Improvements: Encodings

jonkv@ida

- Suppose we have 4 robots, 10 locations
  - Current action representation: move(robot, from, to)
    - 4*10*10 = 400 instances = 400 propositions for the SAT solver to handle (per step in the plan!)
  - One alternative representation (others in the book!):
    - **move**(robot): 4 propositions
    - **movefrom**(robot, from): 40 propositions
    - **moveto**(robot, to): 40 propositions
    - Total: 84 propositions
  - Requires different axiom encodings!

- Many other improvements have been made
  - But we're focusing on the primary ideas behind SAT planning

Improvements: Performance

jonkv@ida

- Recall that we can find *binary mutexes* in planning problems
  - Example: If $h_2(p, q) = \infty$,
    then p and q are not true "together" in any reachable state
  - Can provide the formula *not (p and q)* to the solver
    - If p is made true, the solver can *immediately* conclude q is false
    - Reduces the search space
- Other forms of *state constraints* are also useful

Example SAT Planners

SATPlan and BlackBox

jonkv@ida

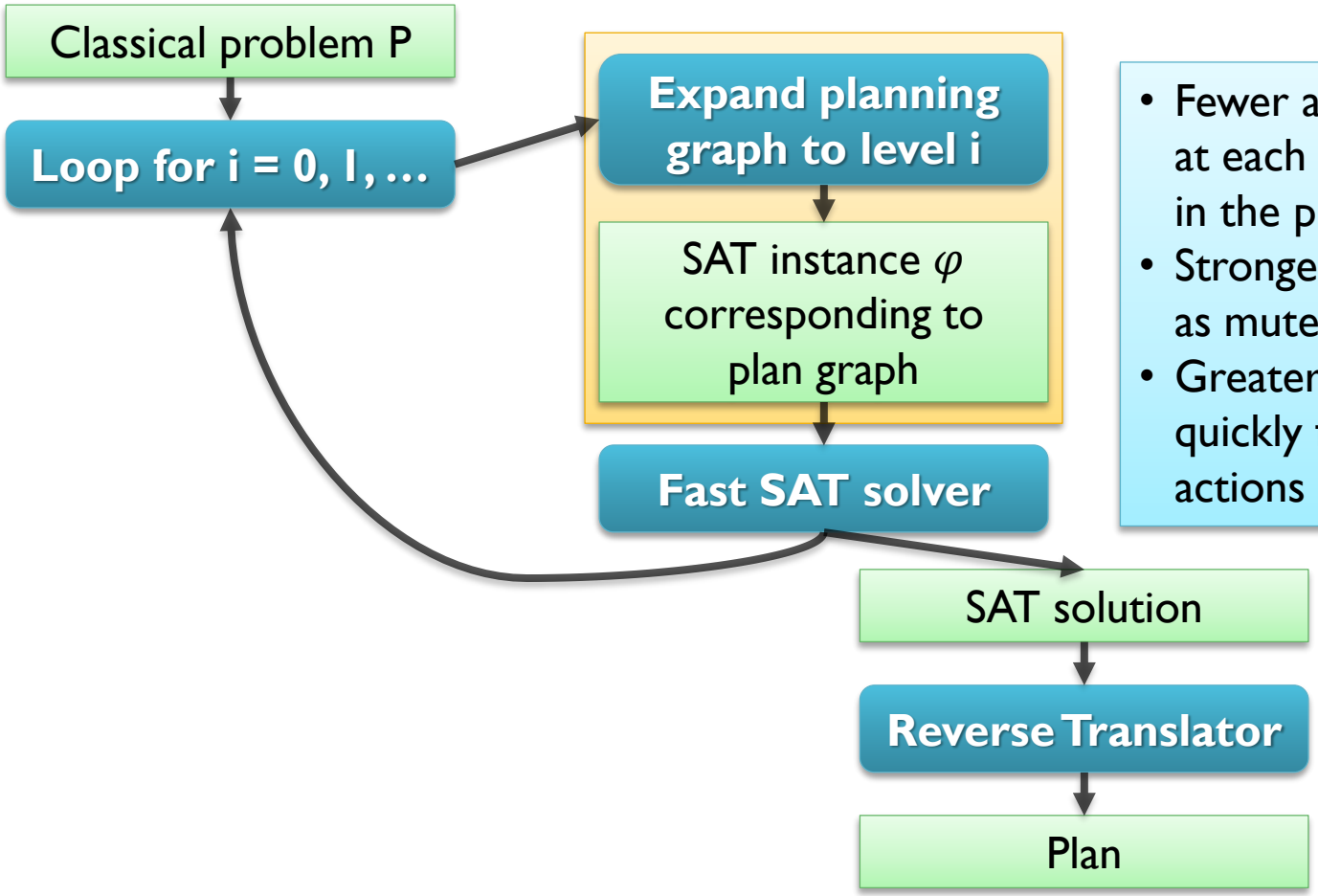
- Pioneered by SATPLAN (Kautz & Selman, 1992)

- Idea behind BlackBox (Kautz & Selman, 1999):
  - SAT planning has several similarities to GraphPlan
    - Both frameworks use iterative deepening
    - Both have two phases
      - Creating a specific representation, and then searching it
      - GraphPlan: Create a **plan graph**, then regression search
      - SAT planning: Create a set of clauses, then apply a SAT solver's search alg.

The BlackBox Planner (2)

jonkv@ida

- BlackBox:
  - Uses the GraphPlan version of parallel plans: Sequence of sets of actions
  - Requires a different encoding, but the same basic ideas apply



- Fewer actions to consider at each step – only those in the plan graph
- Stronger constraints, such as mutexes
- Greater opportunities to quickly find inapplicable actions

Performance

jonkv@ida

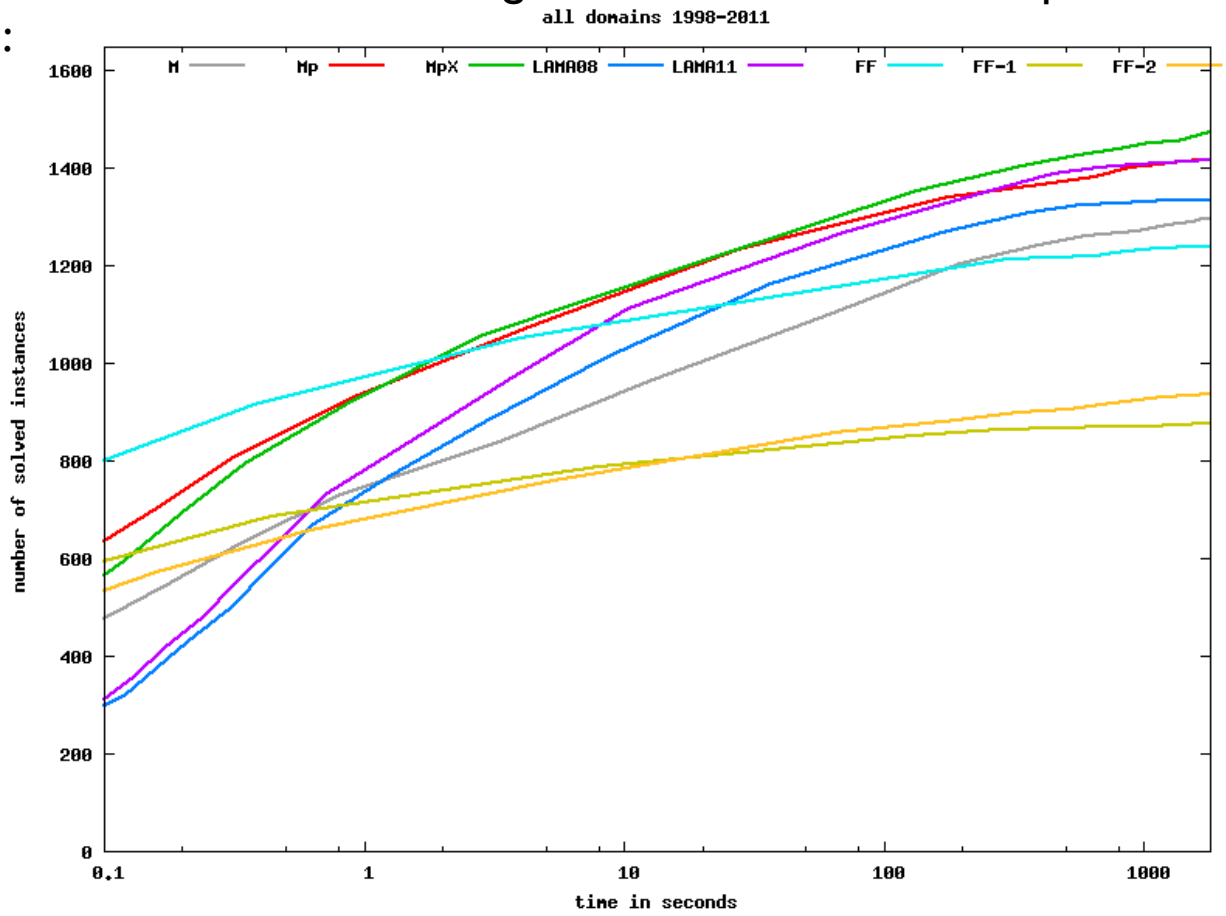
- Performance of BlackBox / SATplan in planning competitions:
  - **1998-2000**: Satisficing planning (find any plan)
    - 1998: Competitive
    - 2000: Other planners had improved
  - **2004-2006**: Optimizing planning (find the shortest plan)
    - 2004: First place
    - 2006: Tied for first place with MAXPLAN, a variant of SATplan
  - Small change in modeling + huge improvements in SAT solvers!

wff	vars	clauses	sato 1997	satz 1997	zChaff 2001	jerusat 2003	siege 2003	MiniSat 2005
p05	3,656	31,089	13.23	0.61	0.01	0.01	0.01	0.02
p15	10,671	143,838	x	4.85	0.05	0.13	0.03	0.09
p18	34,325	750,269	x	x	13.92	6.59	4.85	2.55
p20	40,304	894,643	x	x	14.75	10.35	8.68	10.03
p28	249,738	13,849,105	x	x	846.72	79.59	12.74	27.80

Madagascar

jonkv@ida

- Newer example: **Madagascar** (Jussi Rintanen)
  - Many extensions and improvements upon these general ideas
  - Competitive performance
    - Some problems in IPC-2011 – according to the author's own example (M, Mp, MpC):



http://users.ics.aalto.fi/rintanen/jussi/papers/Rintanen14IPC.pdf