Automated Planning

Planning by Translation to Propositional Satisfiability

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

Background

- Propositional satisfiability (SAT):
  - Let ϕ be a propositional formula
    - For example, (walking → alive) ∧ (alive → breathing) ∧ (breathing)
  - Is there a **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)

- 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)

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

Make use of all these efficient algorithms…

Running Example

- 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
    - 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, l) ^ ¬ at(b, l)
- Corresponding **problem instance**:
  - one robot: rob1
    one box: box1
    two locations: loc1, loc2

Key Ideas

Key Ideas (1)

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

"Action propositions": Which actions are executed, and when?

"Fact propositions": Which facts are true, and when?

Assignments	…	…	…	…	…	…	
→	–	–	–	–	–	–	
→	–	–	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)

- 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	–	
Seq. plan!	–	true	true	–	–	true	**true**	
	true	–	–	true	true	–	–	
	true	–	true	true	true	–	–	
Seq. plan!	true	true	–	–	true	true	**true**	
	true	true	true	true	true	true	–	

Encoding State Sequences using State Propositions

Propositionalization

- SAT solvers 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)
  - Becomes a proposition: at-rob1-loc1
- **__Instantiate__** all operators to 0-param **__actions__**
  - A first-order operator: move(robot r, loc l, loc l')
  - Becomes many actions: move-rob1-loc1-loc1, move-rob1-loc1-loc2, …

Similar to the **__set-theoretic__** classical representation

__Looks__ as if we still have parameters…

To the solver, at-rob1-loc1 could as well be called prop53280!

Multiple States

- But **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…

at-rob1-loc1
at-rob1-loc2
…

The planner keeps track of distinct states

at-rob1-loc1: true
at-rob1-loc2: false
c-rob1-box1: false
…

pickup-rob1-box1-loc1

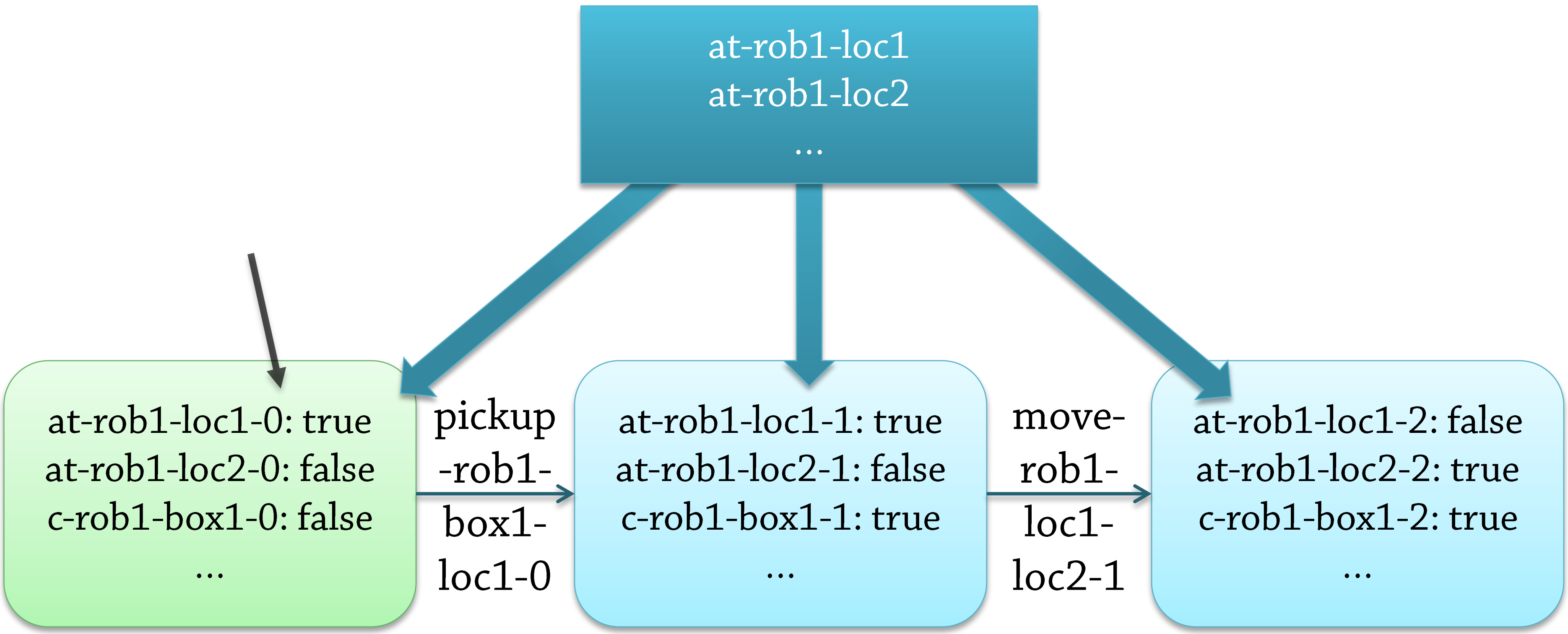
at-rob1-loc1: true
at-rob1-loc2: false
c-rob1-box1: true
…

move-rob1-loc1-loc2

at-rob1-loc1: false
at-rob1-loc2: true
c-rob1-box1: true
…

Multiple States (2)

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



Multiple States (3)

- Now we can view a sequence of states as a single assignment

at-rob1-loc1
at-rob1-loc2
…

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
…

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

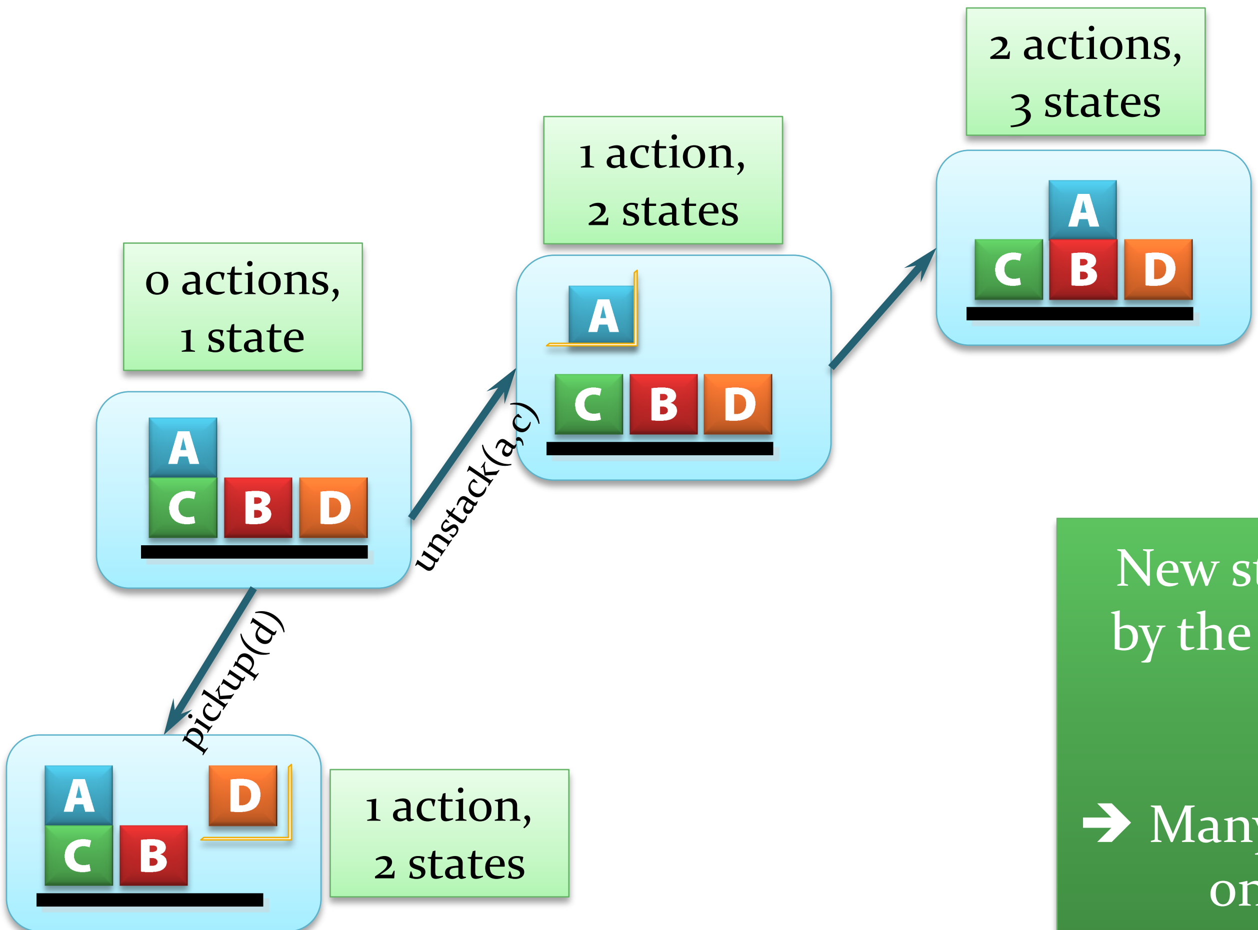
Bounded Planning

Observation

- Observation:
  - Our example problem has 5 atoms
    - at(rob1,loc1)
      at(rob1,loc2)
      at(box1,loc1)
      at(box1,loc2)
    - carrying(rob1,box1)
  - Each **SAT assignment** should contain…
    - The truth value of each atom **in each state**
    - With *n* states, we need 5*n propositions
    - What is the value of *n*?

Solution Length

- But 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" by the planning algorithm as needed

→ Many "copies" of on(A,B), one for each state

Bounded Planning

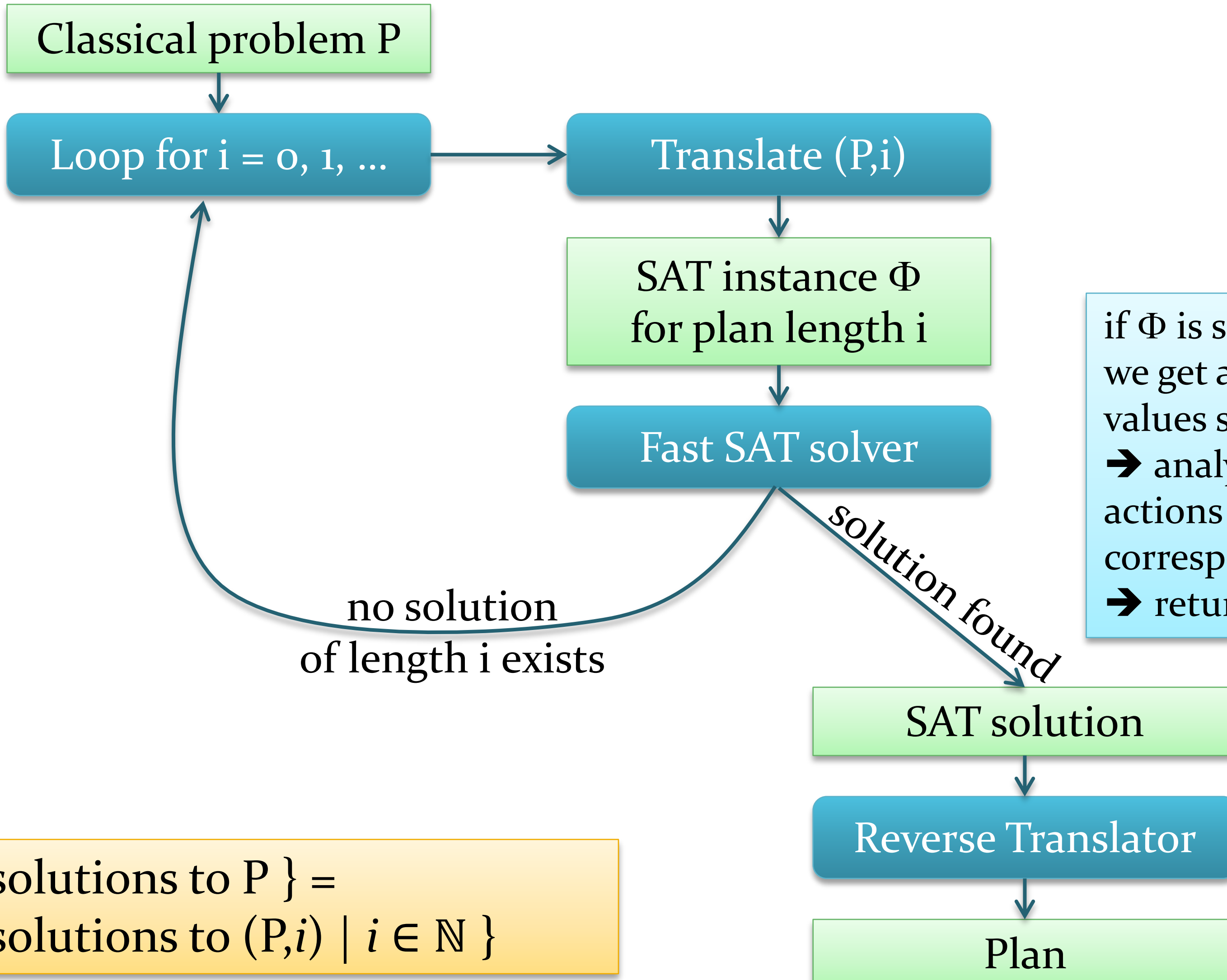
- Each SAT problem has a **fixed number of propositions**
  - ➔ Can't expand "storage" indefinitely, as forward state space planners do

- Solution: Use the SAT solver for **bounded** planning!

> A solution to the **bounded** planning problem (P,n)
> is a solution of length *n*
> to the **classical** planning problem P

Iterative Search

- Use a form of **<u>iterative deepening</u>** 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) | $i \in \mathbb{N}$ }

Remaining Problem

- Remaining problem to solve:
  - Using propositional satisfiability
    to find a plan 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

- At this point, we have no formulas!
  - **Every** SAT assignment is a solution…

Let us view an assignment as "state-based",
even though the SAT solver only sees a single set of propositions…

	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*
	32 combinations of possible values	32 combinations of possible values		32 combinations of possible values

32^(n+1) combinations of possible values,
each of which is a SAT solution

Formulas in Φ: Initial State

- We begin by defining the **initial state**
  - Notation:
    - L = { **all** atoms in the 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:
      at-rob1-loc1-0 ∧ at-box1-loc2-0 ∧
      ¬at-rob1-loc2-0 ∧ ¬at-box1-loc1-0 ∧
      ¬carrying-rob1-box1-0
  - General formula:
    - $$\bigwedge \{\, atom_0 \mid atom \in s_0 \} \wedge$$
      $$\bigwedge \{ \neg atom_0 \mid atom \in L - s_0 \,\}$$

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

- Now **only** assignments satisfying the initial state formula are 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*
	Completely defined 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 Fluents

- Satisfiability has no concept of "**finding actions**"!
  - Solution: Use additional propositions to 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*!

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

Representation Overview

	Time 0	Time 1		Time *n*
Fact propositions	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 propositions	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

- 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
    - ...

Representation Overview

	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

- For **<u>every action</u>** *a* and **<u>every timepoint</u>** $i < n$:
  - **<u>If</u>** the precondition of *a* is not true in state *i*, then *a* cannot be executed at step *i*
    - precond(a) false in state *i* ➔ *a* not executed in step *i*
    - Logically equivalent: *a* executed in step *i* ➔ precond(a) true in state *i*
  - Formula:
    - $a_i \Rightarrow \bigwedge \{ p_i \mid p \in \text{precond}(a) \}$
  - There are **<u>SAT assignments</u>** where:
    - precond(a) is false in state i
    - a is executed in step i
  - But these assignments do not satisfy all formulas ➔ are not **<u>solutions</u>**

Formulas in Φ: Action Effects

- 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:
    - $a_i \Rightarrow \bigwedge \{ e_{i+1} \mid e \in \text{effects}(a)\}$

Formulas in Φ: Actions (2)

$$a_i \Rightarrow \bigwedge \{p_i \mid p \in \mathrm{precond}(a)\} \wedge \bigwedge \{e_{i+1} \mid e \in \mathrm{effects}(a)\}$$

- 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

	Time 0	Time 1	
	Specified by initial state		
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	Values defined by effects These should remain unchanged

Preconds

effects

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

Assume **move-rob1-loc1-loc2-0** is true

move-rob1-loc1-loc2-0 $\Rightarrow$

precond $\wedge$

at-rob1-loc2-1 $\wedge$ $\neg$at-rob1-loc1-1

Enough?

- 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 a **cause**.

  - $\neg$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

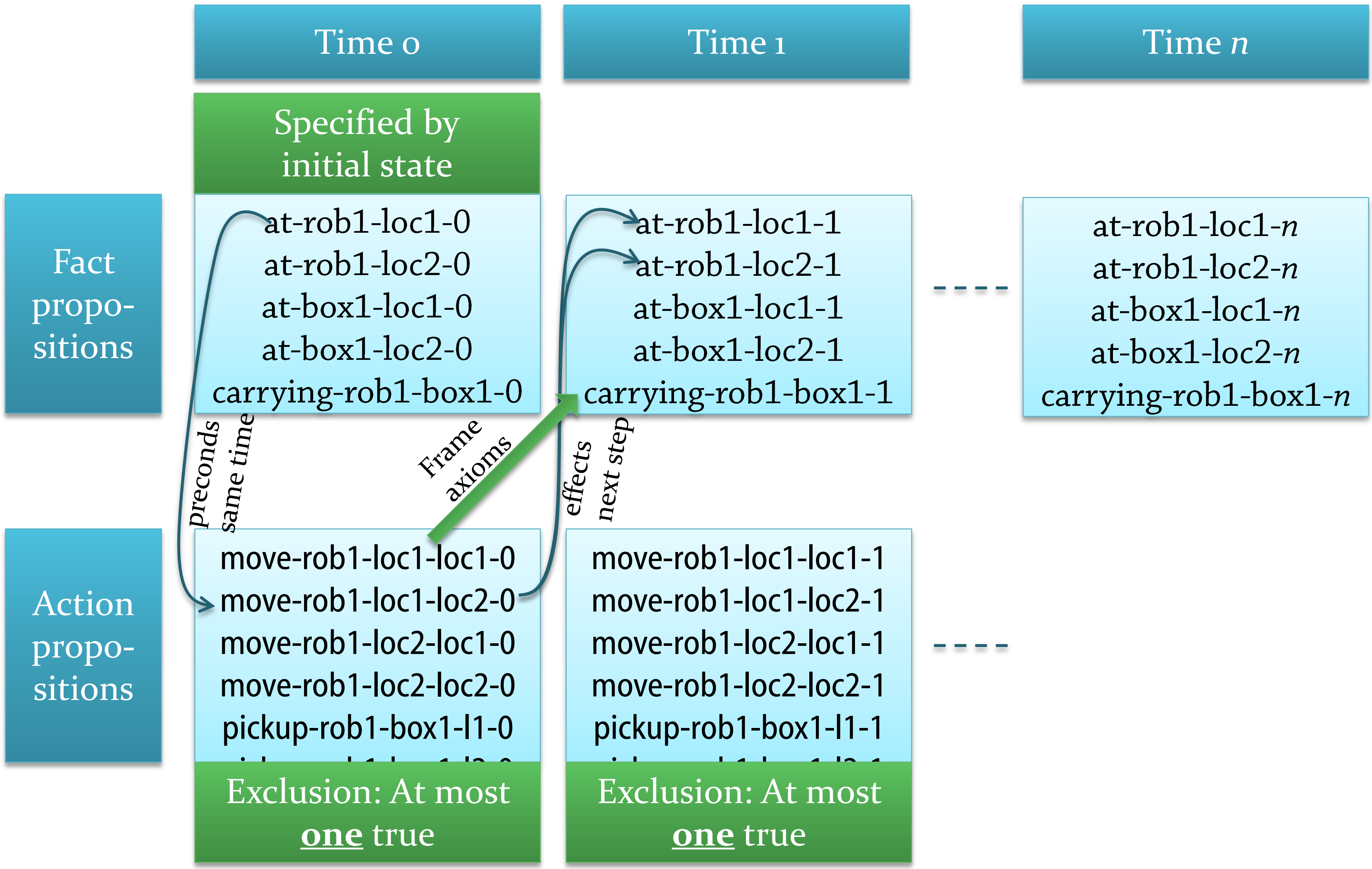
- **<u>Explanatory frame axioms:</u>**
  - One formula for every atom l and every timepoint $i < n$
  - If l **<u>changes</u>** to **<u>true</u>** between s_i and s_{i+1},
    then the action at step i must be responsible:

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

In general there may be more than one possible cause ➔ a **<u>disjunction</u>** to the right of ⇒

Example:
¬at-me-loc1-0 ∧
at-me-loc1-1 =>
walk ∨ run ∨ drive

Representation Overview



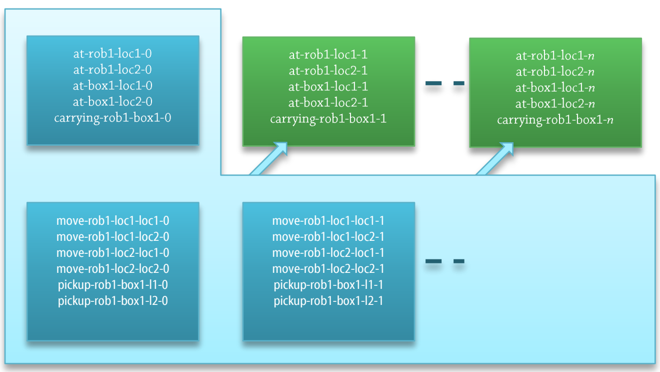
Finding Solutions of Fixed Length

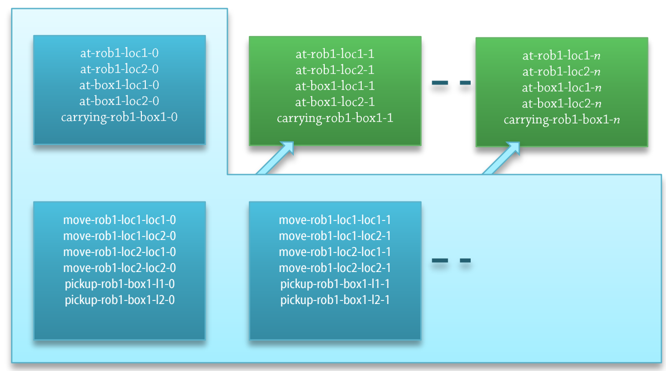
Finding Solutions

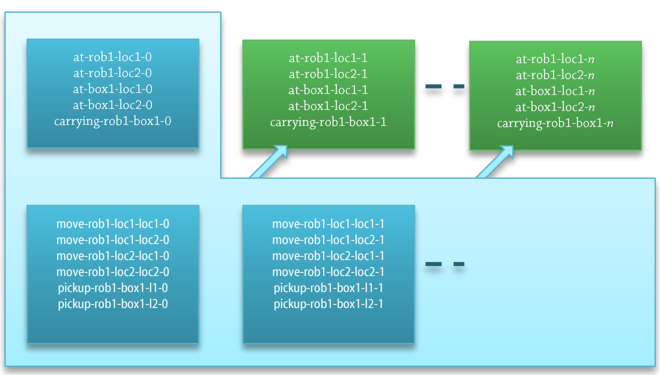
- If we use the current encoding for the problem (P,n):
  - We have one **SAT solution**
    for every **executable action sequence** of length n
    - Some of these may satisfy the goal
    - Some of them may not
  - We want one **SAT solution**
    for every **solution plan** of length n
    - Should keep only those SAT solutions
      where the final state satisfies the goal

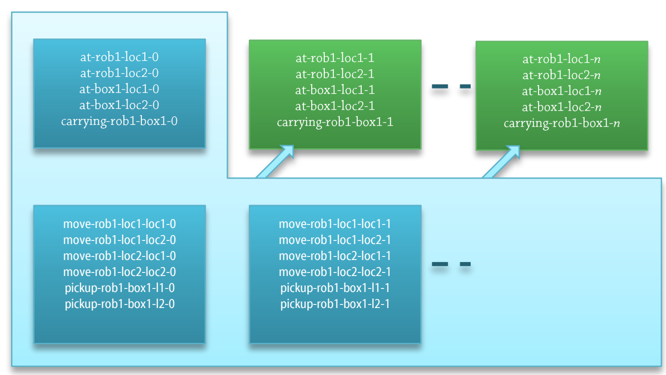
Executable Action Sequences

- Suppose you have 4 SAT solutions for the current formulas
  - Each one corresponds to an executable action sequence



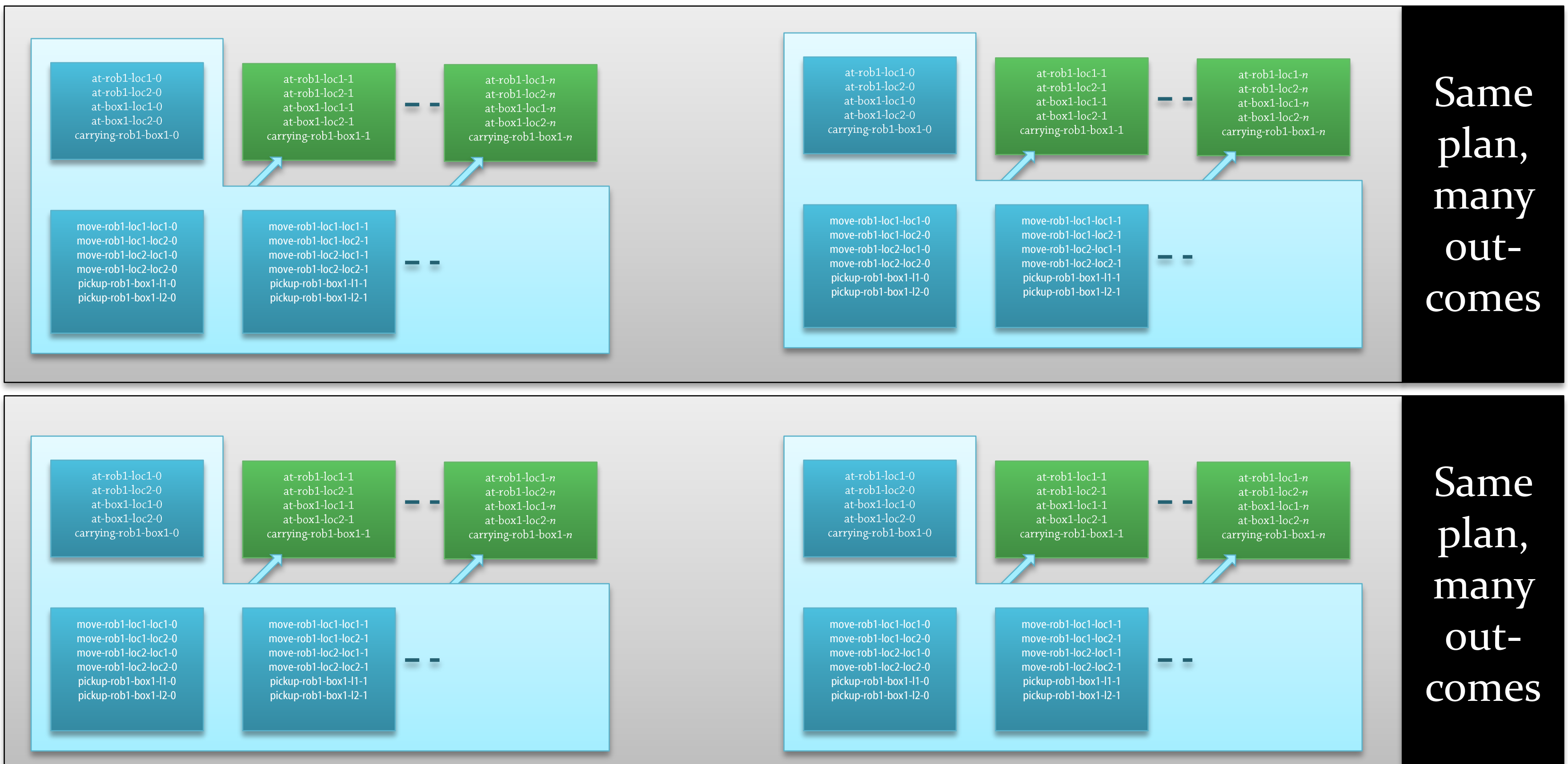






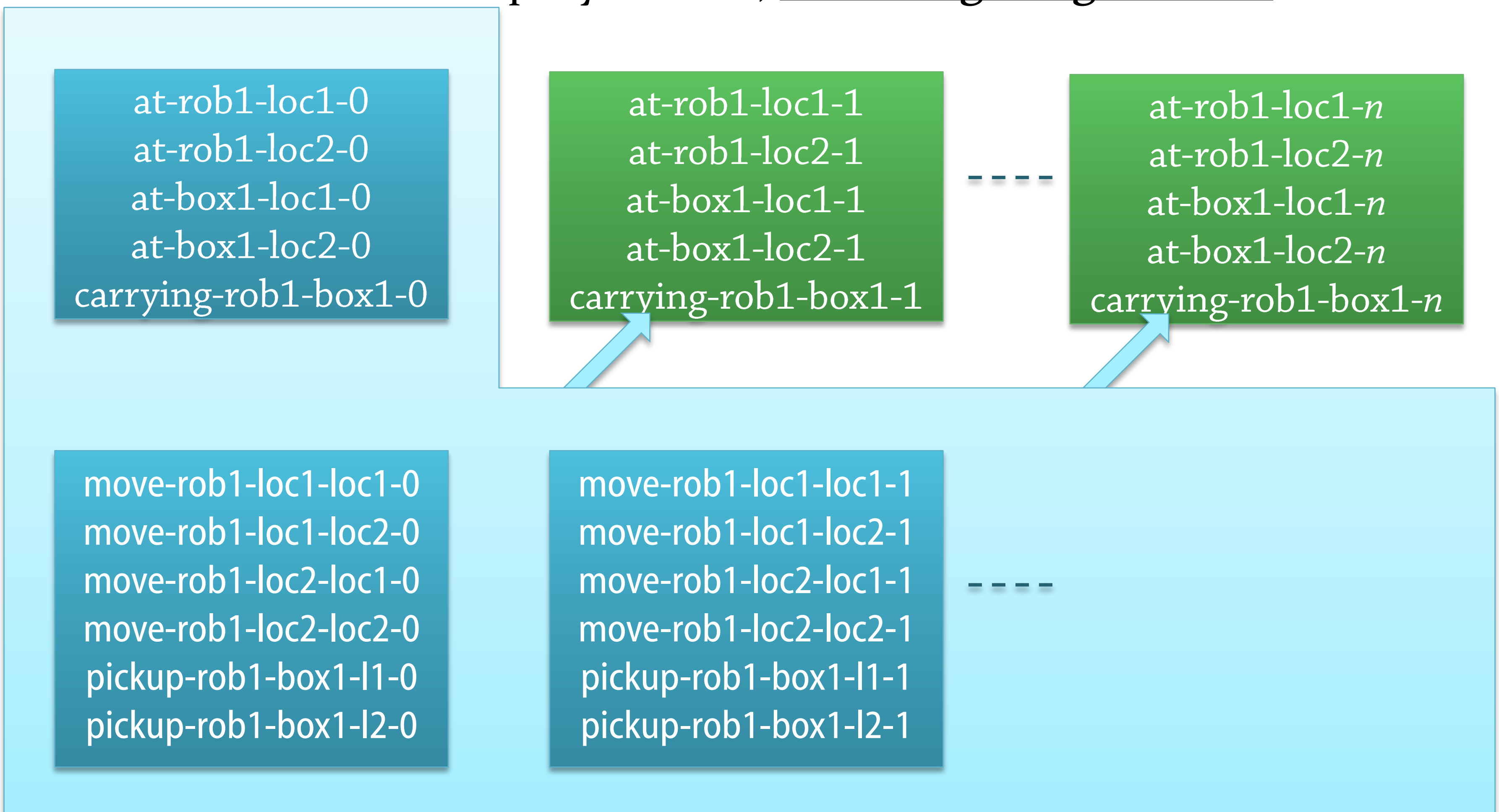
Different Executable Sequences

- If we allowed nondeterministic actions, incomplete states
  - **One** plan could lead to **many** different outcomes
    - Many SAT solutions with the same plan
    - Generate **all** solutions, group them – check if **all** outcomes satisfy the goal



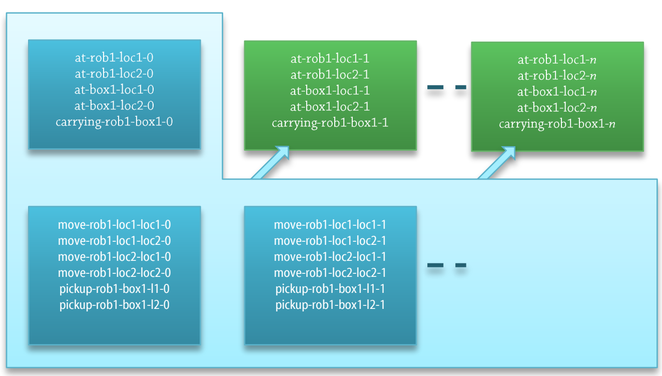
Completely Defined States

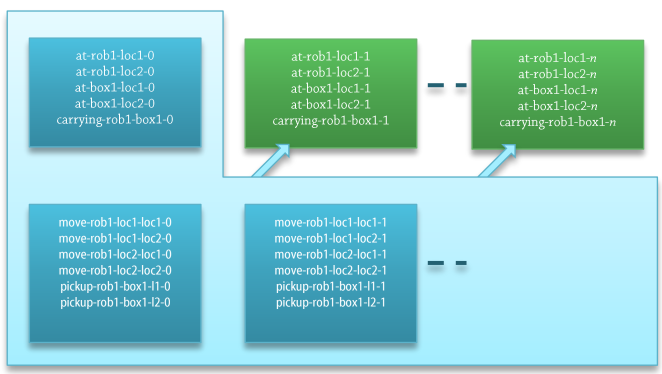
- In deterministic planning:
  - Given an initial state and an assignment to action propositions, all other states are uniquely defined, **<u>including the goal state</u>**

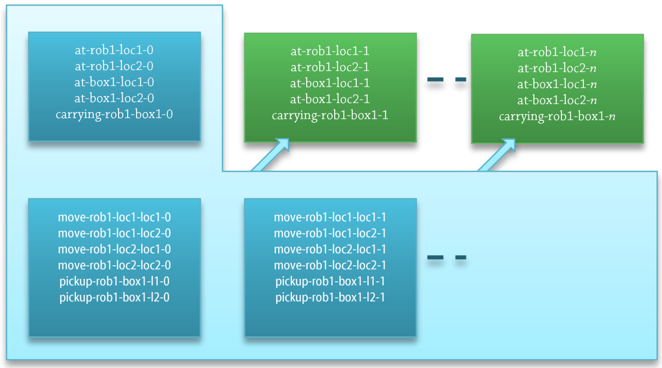


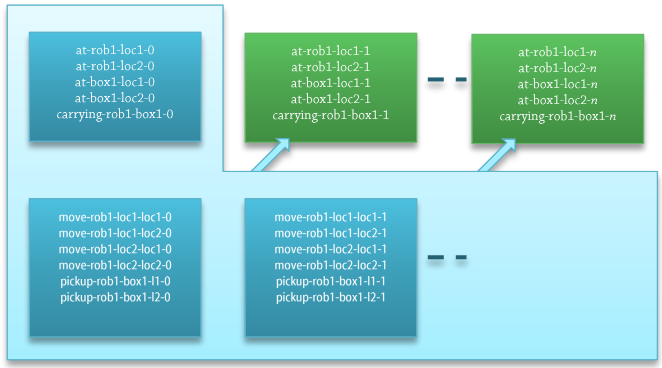
Different Executable Sequences

- Given determinism:
  - Each SAT solution
    must correspond to a **<u>different</u>** executable action sequence



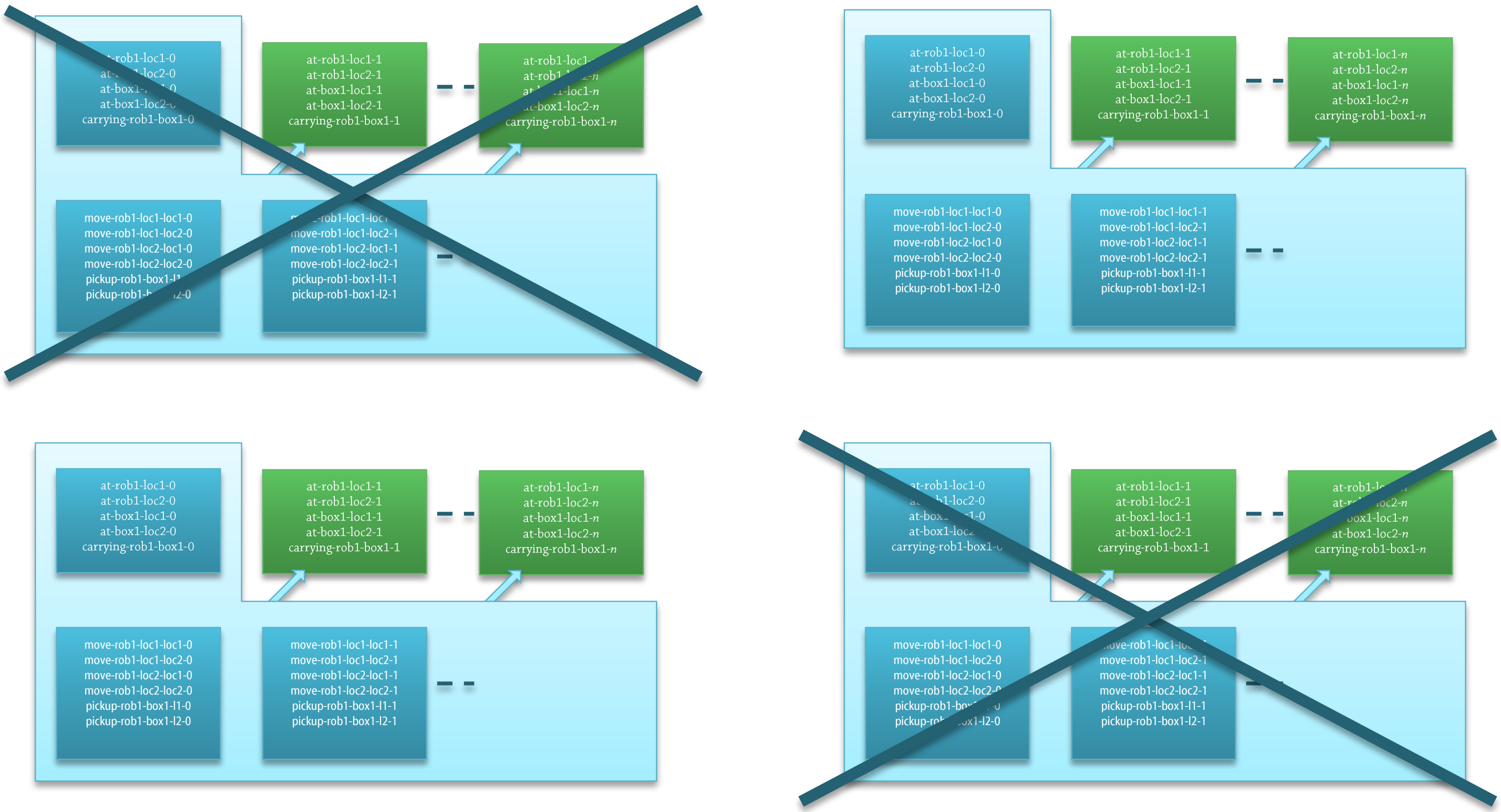






Solution Plans

- Remove those where the last state does not satisfy the goal
  - ➔ All of the remaining ones correspond to solution plans



Formulas in Φ: Goal

- Therefore we can **keep** all solutions satisfying the goal:
  - Simply by **claiming** that the goal formula is true
  - $\bigwedge\{lit_n \mid lit \in g^+\} \wedge$
    $\bigwedge\{\neg lit_n \mid lit \in g^-\}$,
    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

	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	- - - -	

Example

Creating a Single-Step Plan

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	
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

Creating a Single-Step Plan (2)

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!

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)

Initial state
at-rob1-loc1-0 ∧
¬at-rob1-loc2-0 ∧
¬carrying-rob1-box1-0 ∧
…

Action axioms
pickup-rob1-box1-loc1-0 ⇒
at-rob1-loc1-0 ∧
at-box-loc1-0 ∧
¬at-box-loc1-1 ∧
carrying-rob1-box1-1, …

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	false
move-rob1-loc2-loc1-0	false
move-rob1-loc2-loc2-0	false
pickup-rob1-box1-l1-0	true
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	true

Additional backtracking…

Creating a Single-Step Plan (4)

Initial state

at-rob1-loc1-0 ∧
¬at-rob1-loc2-0 ∧
¬carrying-rob1-box1-0 ∧
…

Action axioms

pickup-rob1-box1-loc1-0 ⇒
at-rob1-loc1-0 ∧
at-box-loc1-0 ∧
¬at-box-loc1-1 ∧
carrying-rob1-box1-1, …

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	false
move-rob1-loc2-loc1-0	false
move-rob1-loc2-loc2-0	false
pickup-rob1-box1-l1-0	true
pickup-rob1-box1-l2-0	false

at-rob1-loc1-1	true
at-rob1-loc2-1	false
at-box1-loc1-1	false
at-box1-loc2-1	false
carrying-rob1-box1-1	true

Consistent!

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) ∧ …

Advantages?

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

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	
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	true

Frame Axioms

(¬carrying-rob1-box1-0 ∧ carrying-rob1-box1-1 ⇒
pickup-rob1-box1-l1-0 ∨ pickup-rob1-box1-l2-0) ∧ …

Concurrent Planning?

Formulas in Φ

- SAT planning can be used to generate **concurrent** plans
  - The solver can make many action fluents true at the same time step, without making the model inconsistent

	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*
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	- - - -	

Several of these can be true!

Formulas in Φ

- 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
  - This is **logically consistent**
    but results in a plan where we are at two places at the same time
- We must tell the SAT solver that this is not intended!
  - Not covered in this course

Discussion

Improvements and Extensions

- 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(from): 10 propositions
    - moveto(to): 10 propositions
    - Total: 24 propositions
  - Requires different axiom encodings!

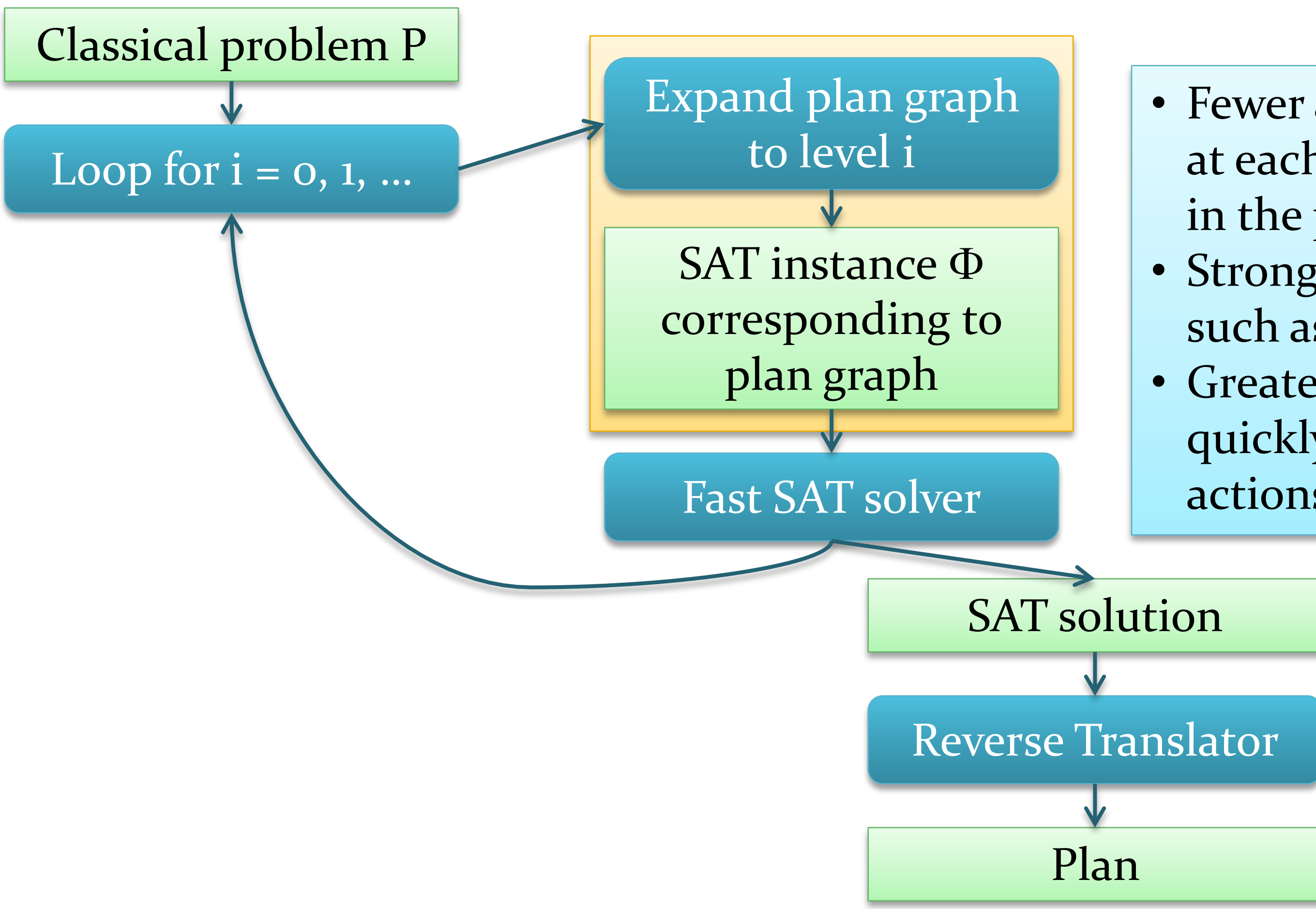
- Many other improvements have been made
  - But we're focusing on the primary ideas behind SAT planning

The BlackBox Planner

- 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)

- Idea behind BlackBox planner
  - 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

- Performance of BlackBox / SATplan in planning competitions:
  - 1998-2002: Satisficing planning (find any plan)
    - 1998: Competitive
    - 2000: Other planners had improved
    - 2002: Did not participate
  - 2004-2011: Optimizing planning (find the shortest plan)
    - 2004: First place
    - 2006: Tied for first place with MAXPLAN, a variant of SATplan
    - 2008: Did not participate
    - 2011: Did not participate
  - 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