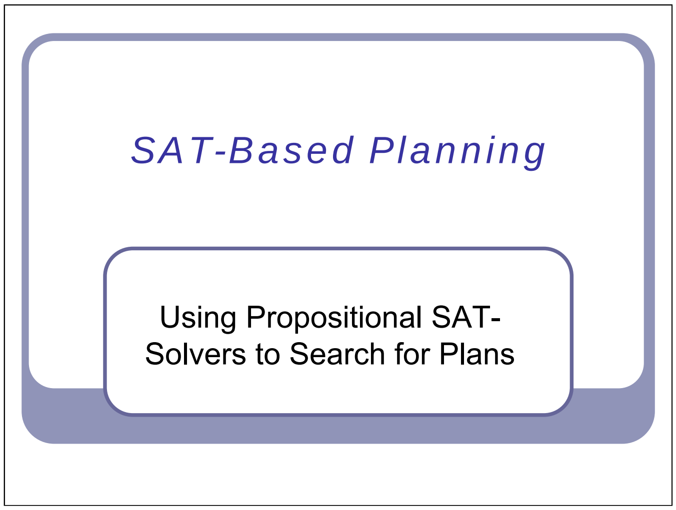
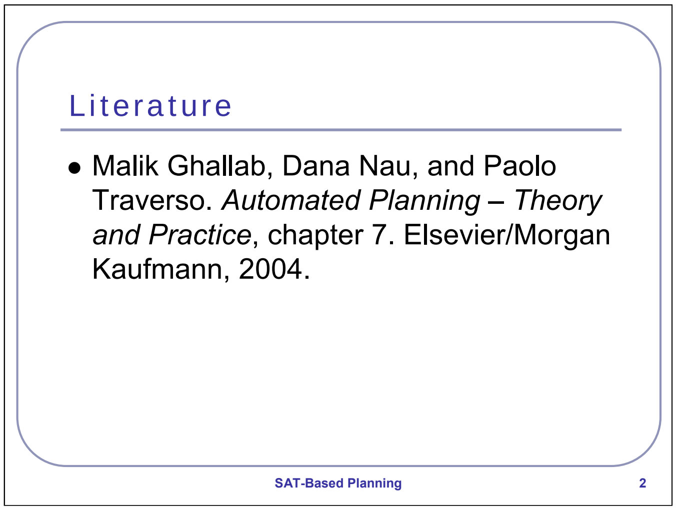


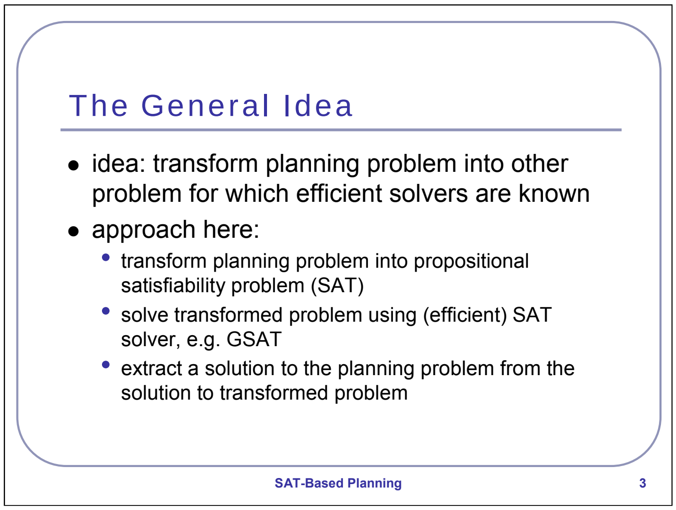
SAT-Based Planning

•Using Propositional SAT-Solvers to Search for Plans



Literature

- **Malik Ghallab, Dana Nau, and Paolo Traverso. *Automated Planning – Theory and Practice*, chapter 7. Elsevier/Morgan Kaufmann, 2004.**



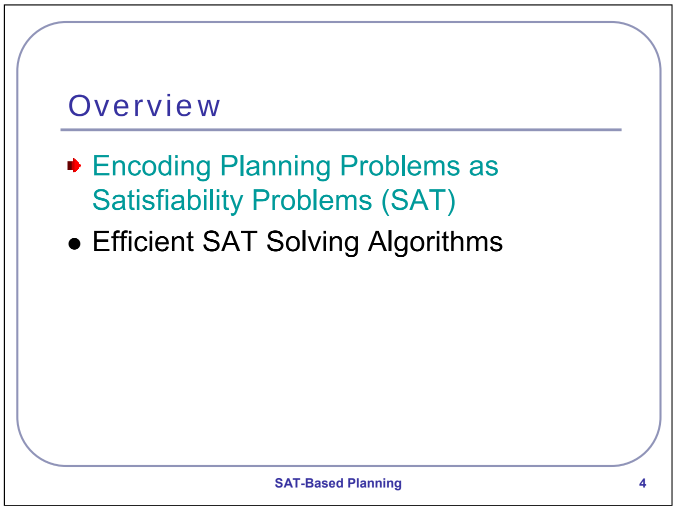
The General Idea

•idea: transform planning problem into other problem for which efficient solvers are known

•approach here:

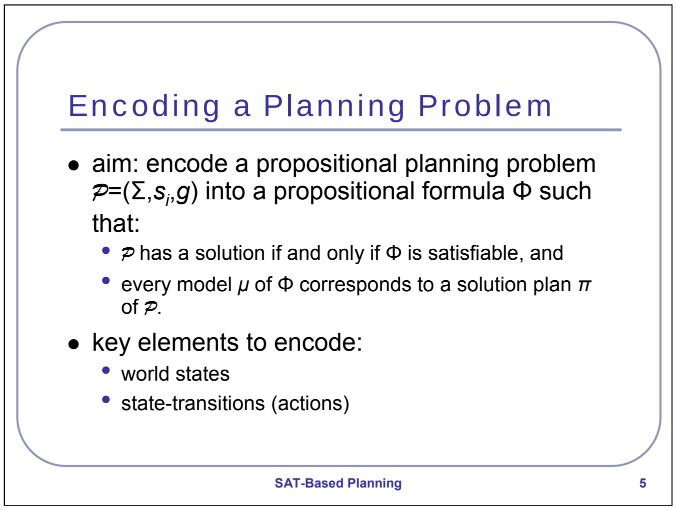
- **•transform planning problem into propositional satisfiability problem (SAT)**
  - •SAT: problem of determining whether a propositional formula is satisfiable (theorem proving)
- **•solve transformed problem using (efficient) SAT solver, e.g. GSAT**
- **•extract a solution to the planning problem from the solution to transformed problem**

•main advantage: exploit recent results in SAT solver for efficient planning



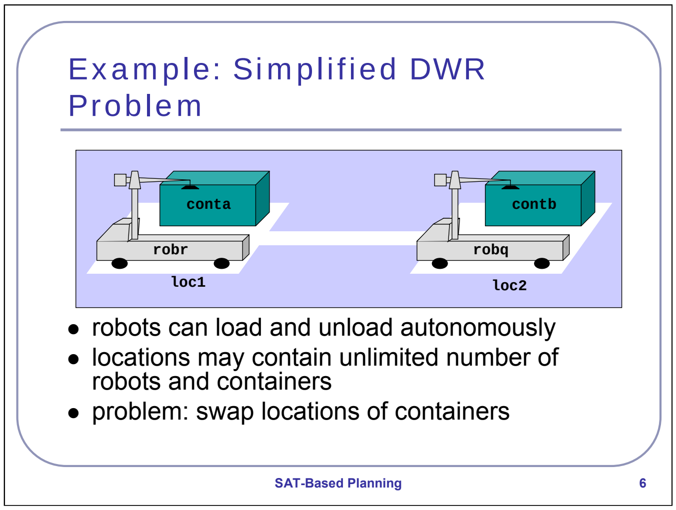
Overview

- **Encoding Planning Problems as Satisfiability Problems (SAT)**
  - now: how to encode a planning problem into SAT problem that has a model iff the planning problem has a solution
- **Efficient SAT Solving Algorithms**



Encoding a Planning Problem

- **aim: encode a propositional planning problem $\mathcal{P}=(\Sigma,s_i,g)$ into a propositional formula Φ such that:**
  - **$\mathcal{P}$ has a solution if and only if Φ is satisfiable, and**
  - **every model μ of Φ corresponds to a solution plan π of $\mathcal{P}$.**
    - model: assignment of truth values to propositions (atoms) in the formula such that formula evaluates to true
    - formula is satisfiable if it has a model (if such an assignment exists)
- **key elements to encode:**
  - **world states**
  - **state-transitions (actions)**
- note: style of encoding determines length of formula and thus difficulty of SAT problem



Example: Simplified DWR Problem

•same problem as for Graphplan

•[figure]

•initial state: as shown

•robots can load and unload autonomously

•locations may contain unlimited number of robots and containers

•problem: swap locations of containers

Simplified DWR Problem: State Proposition Symbols

- robots:
  - *r1* and *r2*: at(robr,loc1) and at(robr,loc2)
  - *q1* and *q2*: at(robq,loc1) and at(robq,loc2)
  - *ur* and *uq*: unloaded(robr) and unloaded(robq)
- containers:
  - *a1*, *a2*, *ar*, and *aq*: in(conta,loc1), in(conta,loc2), loaded(conta,robr), and loaded(conta,robq)
  - *b1*, *b2*, *br*, and *bq*: in(contb,loc1), in(contb,loc2), loaded(contb,robr), and loaded(contb,robq)
- initial state: {*r1*, *q2*, *a1*, *b2*, *ur*, *uq*}

Simplified DWR Problem: State Proposition Symbols

•robots:

•*r1* and *r2*: at(robr,loc1) and at(robr,loc2)

•*q1* and *q2*: at(robq,loc1) and at(robq,loc2)

•*ur* and *uq*: unloaded(robr) and unloaded(robq)

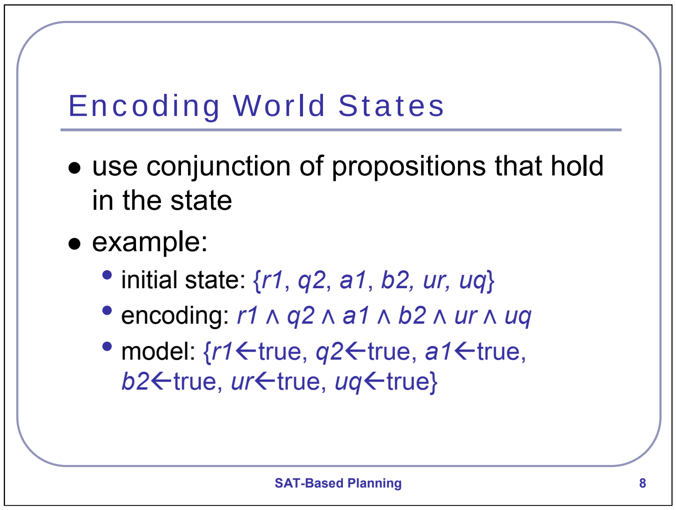
•containers:

•*a1*, *a2*, *ar*, and *aq*: in(conta,loc1), in(conta,loc2), loaded(conta,robr), and loaded(conta,robq)

•*b1*, *b2*, *br*, and *bq*: in(contb,loc1), in(contb,loc2), loaded(contb,robr), and loaded(contb,robq)

•14 state propositions

•initial state: {*r1*, *q2*, *a1*, *b2, ur, uq*}

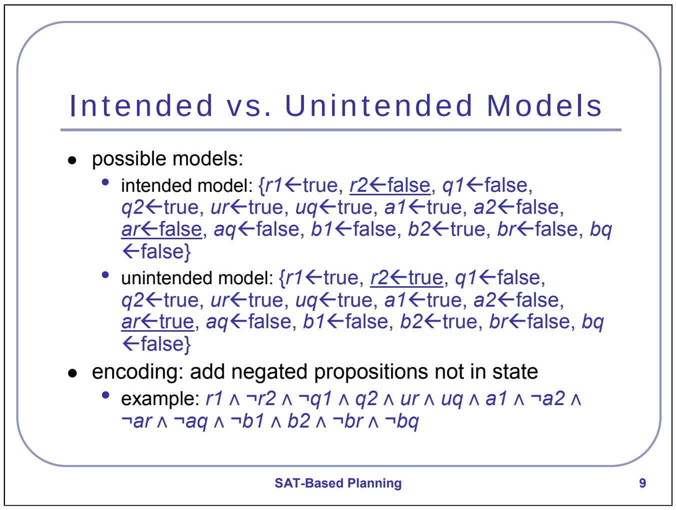


Encoding World States

•use conjunction of propositions that hold in the state

•example:

- **•initial state: {*r1*, *q2*, *a1*, *b2, ur, uq*}**
- **•encoding: *r1* ∧ *q2* ∧ *a1* ∧ *b2* ∧ *ur* ∧ *uq***
- **•model: {*r1*←**true, ***q2*←**true, ***a1*←**true, ***b2*←**true, ***ur*←**true, ***uq*←**true**}**



Intended vs. Unintended Models

•**possible models:**

- •**intended model: {*r1*←**true**, *r2*←false, *q1*←false, *q2*←**true**, *ur*←**true**, *uq*←**true**, *a1*←**true**, *a2*←false, *ar*←false, *aq*←false, *b1*←false, *b2*←**true**, *br*←false, *bq* ←false}**
- •**unintended model: {*r1*←**true**, *r2*←true, *q1*←false, *q2*←**true**, *ur*←**true**, *uq*←**true**, *a1*←**true**, *a2*←false, *ar*←true, *aq*←false, *b1*←false, *b2*←**true**, *br*←false, *bq* ←false}**

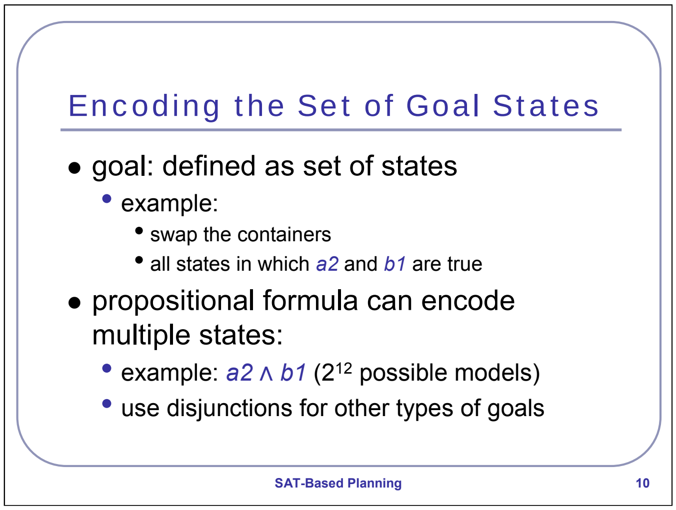
•**encoding: add negated propositions not in state**

- •**example:** $r1 \land \neg r2 \land \neg q1 \land q2 \land ur \land uq \land a1 \land \neg a2 \land \neg ar \land \neg aq \land \neg b1 \land b2 \land \neg br \land \neg bq$

•both models make formula from previous slide true

•differences are underlined

- •unintended model has container a in location 1 and on robot *r* (may be possible, but unintended)
- •unintended model has robot *r* in location 1 and 2 at the same time



Encoding the Set of Goal States

- **goal: defined as set of states**
  - **example:**
    - **swap the containers**
    - **all states in which *a2* and *b1* are true**
- **propositional formula can encode multiple states:**
  - **example: $a2 \land b1$ (2^{12} possible models)**
  - **use disjunctions for other types of goals**
  - other types: set not defined through propositions that must hold, e.g. both containers must be in the same location
- note: every state can be described in this way, allowing for ambiguity

Simplified DWR Problem: Action Symbols

- move actions:
  - Mr12: move(robr,loc1,loc2), Mr21: move(robr,loc2,loc1), Mq12: move(robq,loc1,loc2), Mq21: move(robq,loc2,loc1)
- load actions:
  - Lar1: load(conta,robr,loc1); Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, and Lbq2 correspondingly
- unload actions:
  - Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, and Ubq2 correspondingly

SAT-Based Planning 11

Simplified DWR Problem: Action Symbols

•move actions:

> **•Mr12: move(robr,loc1,loc2), Mr21: move(robr,loc2,loc1), Mq12: move(robq,loc1,loc2), Mq21: move(robq,loc2,loc1)**

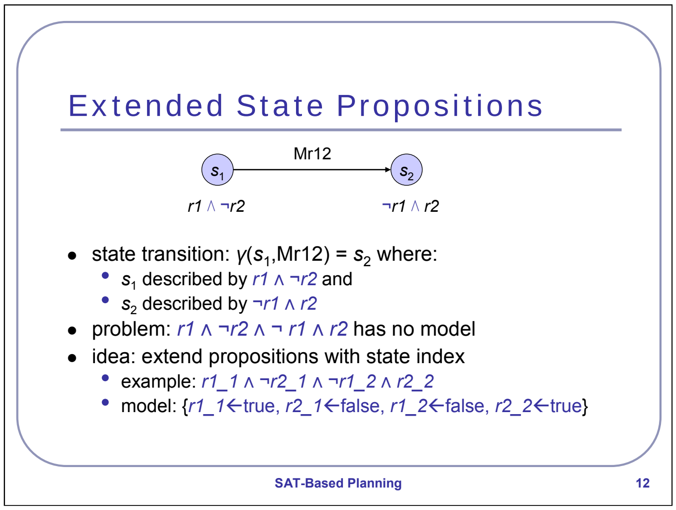
•load actions:

> **•Lar1: load(conta,robr,loc1); Lar2, Laq1, Laq2, Lar1, Lbr2, Lbq1, and Lbq2 correspondingly**

•unload actions:

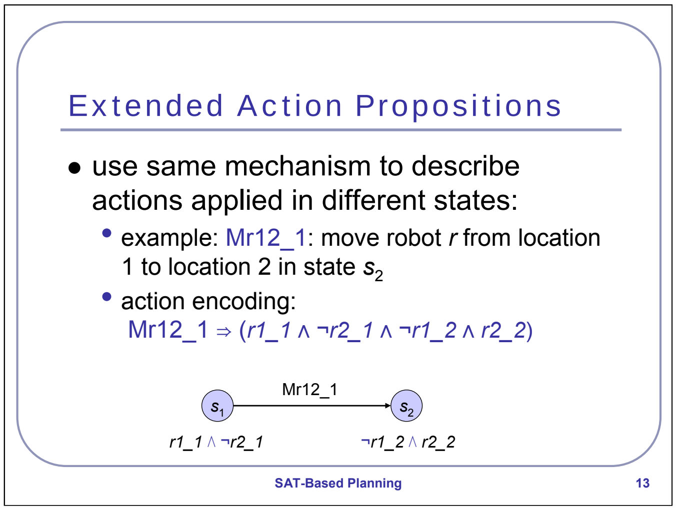
> **•Uar1: unload(conta,robr,loc1); Uar2, Uaq1, Uaq2, Uar1, Ubr2, Ubq1, and Ubq2 correspondingly**

•20 action symbols



Extended State Propositions

- **[figure]**
    - formulas describe state transition when applying Mr12 in state s_1, resulting in s_2
- **state transition: $\gamma(s_1,\text{Mr12}) = s_2$ where:**
    - **s_1 described by $r1 \wedge \neg r2$ and**
    - **s_2 described by $\neg r1 \wedge r2$**
- **problem: $r1 \wedge \neg r2 \wedge \neg\, r1 \wedge r2$ has no model**
- **idea: extend propositions with state index**
    - **example: $r1\_1 \wedge \neg r2\_1 \wedge \neg r1\_2 \wedge r2\_2$**
        - meaning: *r1_1* – robot *r* is at location 1 in state 1
    - **model: {*r1_1*←true, *r2_1*←false, *r1_2*←false, *r2_2*←true}**



Extended Action Propositions

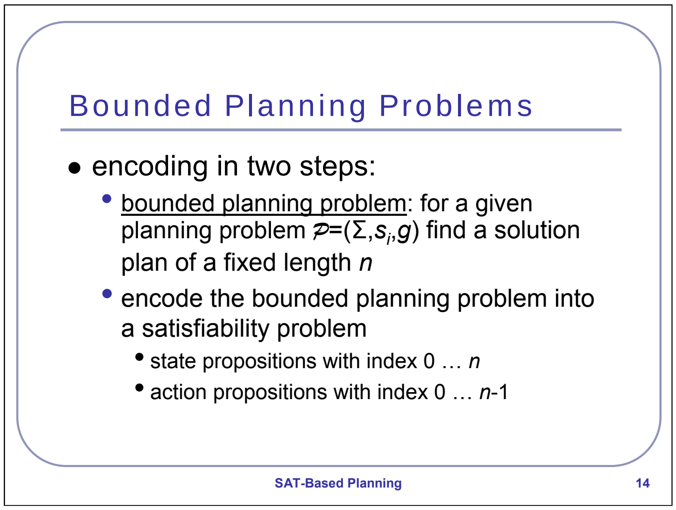
•use same mechanism to describe actions applied in different states:

- **•example: Mr12_1: move robot *r* from location 1 to location 2 in state s_2**
  - •Mr12_1: state index refers to state before that action and has same number; different for planning graph
- **•action encoding: Mr12_1 ⇒ ($r1\_1 \wedge \neg r2\_1 \wedge \neg r1\_2 \wedge r2\_2$)**
  - •model: action must be true; rest remains as before

•[figure]

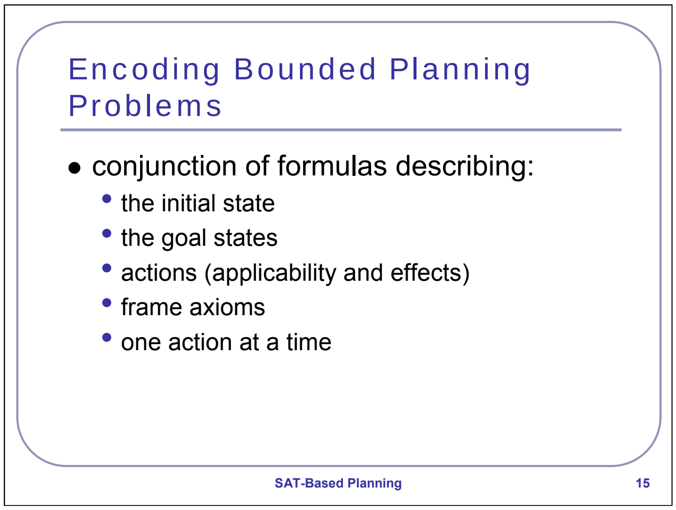
•note:

- •encoding based on same ideas as situation calculus, but propositional
- •encoding of actions as propositional formulas (propositional representation: actions as set-theoretic operators)



Bounded Planning Problems

- **encoding in two steps:**
  - **<u>bounded planning problem</u>: for a given planning problem $\mathcal{P}=(\Sigma,s_i,g)$ find a solution plan of a fixed length n**
    - idea: extend length of bounded planning problem until a plan is found
      - e.g. use binary search on plan length
  - **encode the bounded planning problem into a satisfiability problem**
    - **state propositions with index 0 … n**
    - **action propositions with index 0 … n-1**
- note: if a plan of length $k<n$ exists then it can be extended to a plan of length n with no-op actions

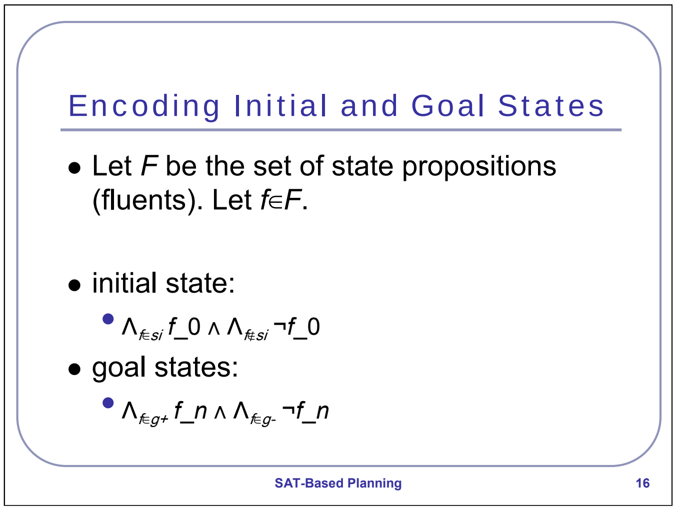


Encoding Bounded Planning Problems

•conjunction of formulas describing:

- **•the initial state**
- **•the goal states**
- **•actions (applicability and effects)**
- **•frame axioms**
- **•one action at a time**

•different types of encodings known: differ in resulting formula length, roughly equivalent to complexity of SAT problem



Encoding Initial and Goal States

- **Let F be the set of state propositions (fluents). Let $f \in F$.**
- **initial state:**
  - $\bigwedge_{f \in si} f\_0 \wedge \bigwedge_{f \notin si} \neg f\_0$
- **goal states:**
  - $\bigwedge_{f \in g+} f\_n \wedge \bigwedge_{f \in g-} \neg f\_n$

Encoding Actions

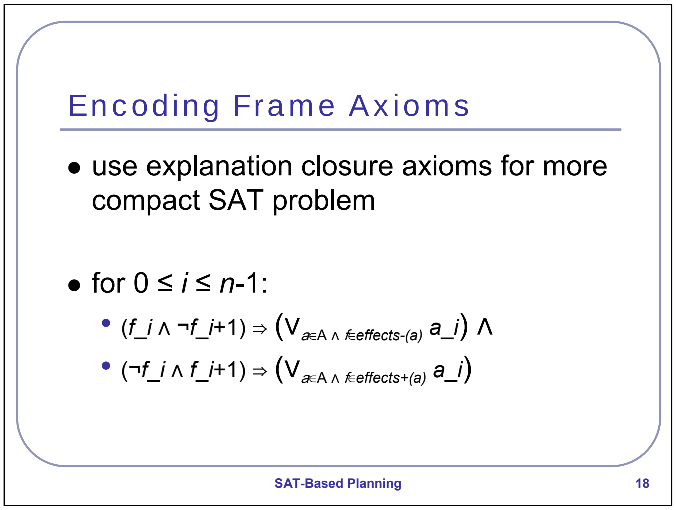
- Let A be the set of action propositions. Let $a \in A$.
- for $0 \leq i \leq n$-1:
  - $a\_i \Rightarrow (\bigwedge_{f \in precond(a)} f\_i \wedge \bigwedge_{f \in effects+(a)} f\_i{+}1 \wedge \bigwedge_{f \in effects-(a)} \neg f\_i{+}1)$

Encoding Actions

•Let *A* be the set of action propositions. Let $a \in A$.

•for $0 \leq i \leq n$-1:

- **$a\_i \Rightarrow (\bigwedge_{f \in precond(a)} f\_i \wedge \bigwedge_{f \in effects+(a)} f\_i{+}1 \wedge \bigwedge_{f \in effects-(a)} \neg f\_i{+}1)$**
- if action *a* is performed in step *i* then:
  - all preconditions must have held before and
  - all positive effects will hold after and
  - all negative effects will not hold after

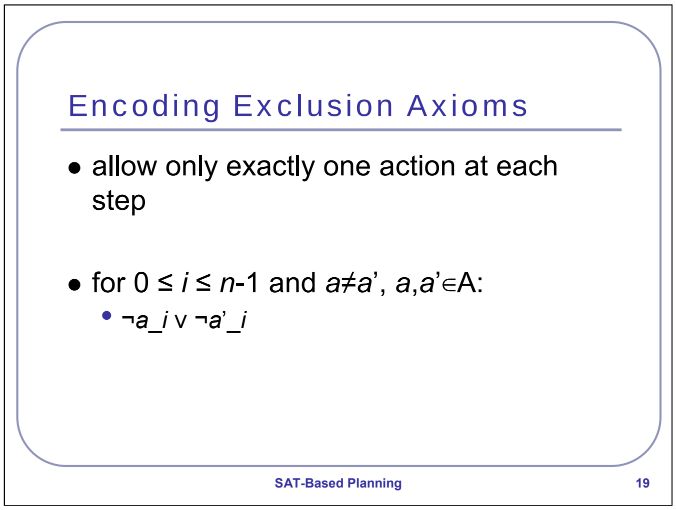


Encoding Frame Axioms

•use explanation closure axioms for more compact SAT problem

•for $0 \leq i \leq n$-1:

- $(f\_i \wedge \neg f\_i{+}1) \Rightarrow (\bigvee_{a \in A \wedge f \in effects\text{-}(a)} a\_i) \wedge$
- $(\neg f\_i \wedge f\_i{+}1) \Rightarrow (\bigvee_{a \in A \wedge f \in effects{+}(a)} a\_i)$



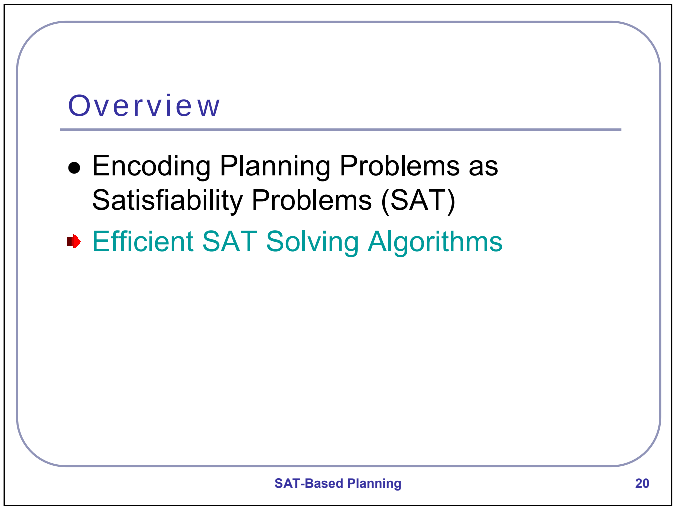
Encoding Exclusion Axioms

•allow only exactly one action at each step

•for 0 ≤ i ≤ n-1 and $a \neq a'$, $a,a' \in$A:

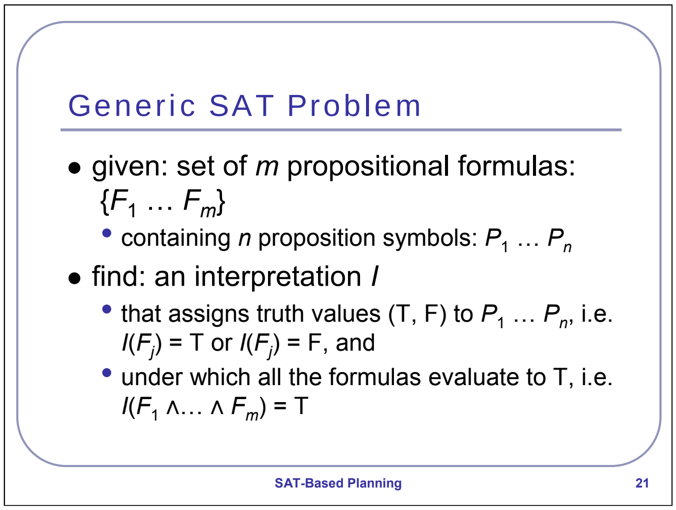
- **$\neg a\_i \vee \neg a'\_i$**

•unlike Graphplan: only one action in each step



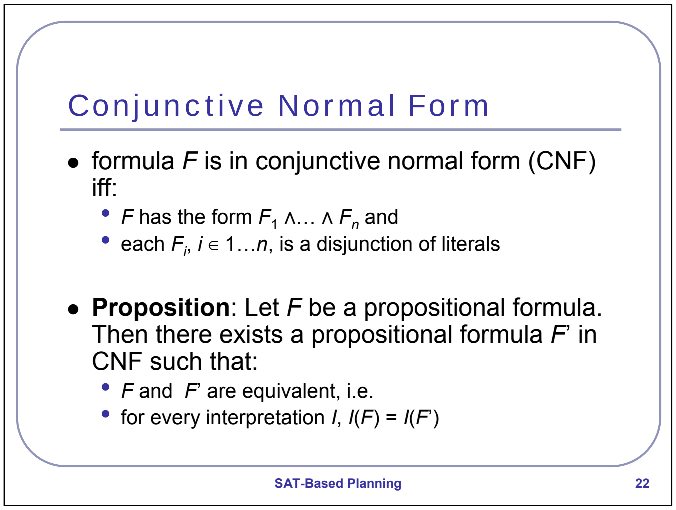
Overview

- **Encoding Planning Problems as Satisfiability Problems (SAT)**
  - just done: how to encode a planning problem into SAT problem that has a model iff the planning problem has a solution
- **Efficient SAT Solving Algorithms**
  - now: efficient algorithms for SAT solving (very quick overview)



Generic SAT Problem

- **given: set of *m* propositional formulas: $\{F_1 \ldots F_m\}$**
    - or one formula that is the conjunction of the formulas in the set
    - **containing *n* proposition symbols: $P_1 \ldots P_n$**
- **find: an interpretation *I***
    - **that assigns truth values (T, F) to $P_1 \ldots P_n$, i.e. $I(F_j)$ = T or $I(F_j)$ = F, and**
    - **under which all the formulas evaluate to T, i.e. $I(F_1 \wedge \ldots \wedge F_m)$ = T**



Conjunctive Normal Form

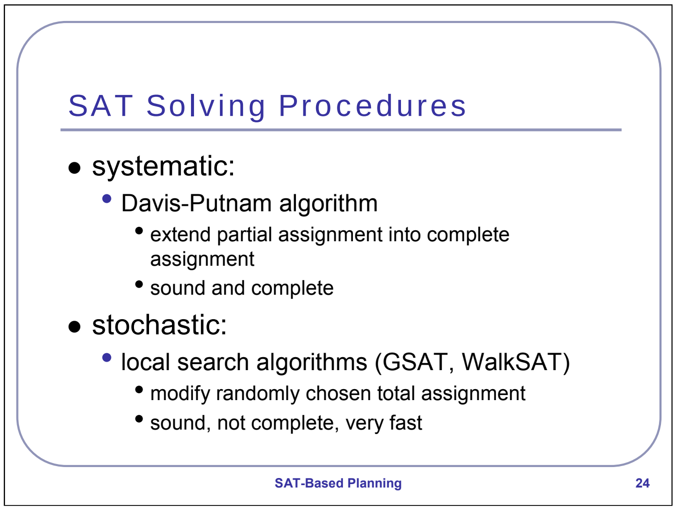
- **formula F is in conjunctive normal form (CNF) iff:**
  - **F has the form $F_1 \wedge \ldots \wedge F_n$ and**
  - **each F_i, $i \in 1 \ldots n$, is a disjunction of literals**
    - disjunction of literals: clause
- **Proposition: Let F be a propositional formula. Then there exists a propositional formula F' in CNF such that:**
  - **F and F' are equivalent, i.e.**
  - **for every interpretation I, $I(F) = I(F')$**

Transformation into CNF

- eliminate implications:
  - $F \leftrightarrow G = F \rightarrow G \wedge G \rightarrow F$
  - $F \rightarrow G = \neg F \vee G$
- bring negations before atoms:
  - $\neg(F \vee G) = \neg F \wedge \neg G$
  - $\neg(F \wedge G) = \neg F \vee \neg G$
  - $\neg(\neg F) = F$
- apply distributive laws:
  - $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$
  - $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$

Transformation into CNF

- **eliminate implications:**
  - $F \leftrightarrow G = F \rightarrow G \wedge G \rightarrow F$
  - $F \rightarrow G = \neg F \vee G$
- **bring negations before atoms:**
  - $\neg(F \vee G) = \neg F \wedge \neg G$
  - $\neg(F \wedge G) = \neg F \vee \neg G$
  - $\neg(\neg F) = F$
- **apply distributive laws:**
  - $F \wedge (G \vee H) = (F \wedge G) \vee (F \wedge H)$
  - $F \vee (G \wedge H) = (F \vee G) \wedge (F \vee H)$



SAT Solving Procedures

•systematic:

- **•Davis-Putnam algorithm**
  - **•extend partial assignment into complete assignment**
  - **•sound and complete**

•stochastic:

- **•local search algorithms (GSAT, WalkSAT)**
  - **•modify randomly chosen total assignment**
  - **•sound, not complete, very fast**

Local Search Algorithms

- basic principles:
  - keep only a single (complete) state in memory
  - generate only the neighbours of that state
  - keep one of the neighbours and discard others
- key features:
  - no search paths
  - neither systematic nor incremental
- key advantages:
  - use very little memory (constant amount)
  - find solutions in search spaces too large for systematic algorithms

Local Search Algorithms

•basic principles:

•keep only a single (complete) state in memory

•generate only the neighbours of that state

•keep one of the neighbours and discard others

•key features:

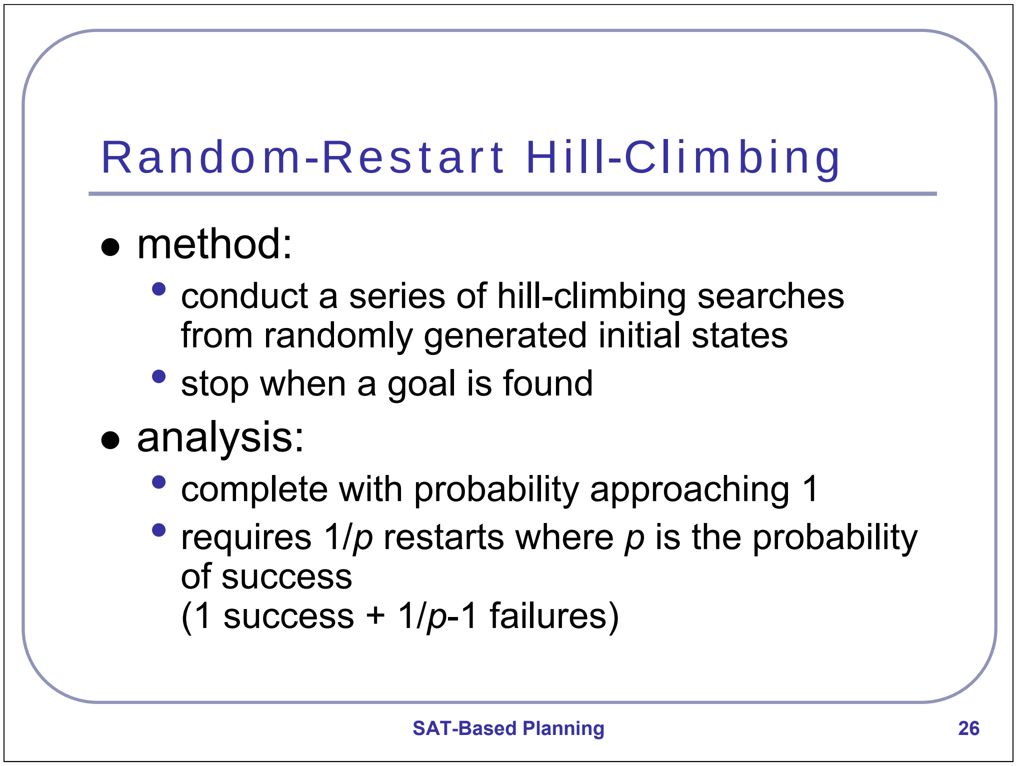
•no search paths

•neither systematic nor incremental

•key advantages:

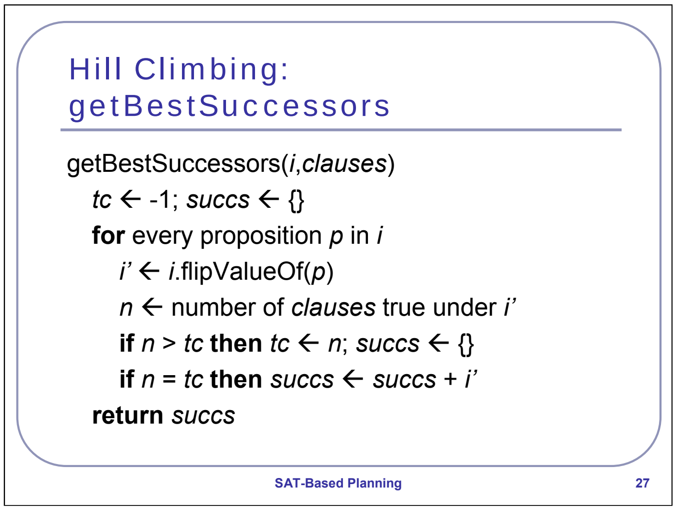
•use very little memory (constant amount)

•find solutions in search spaces too large for systematic algorithms



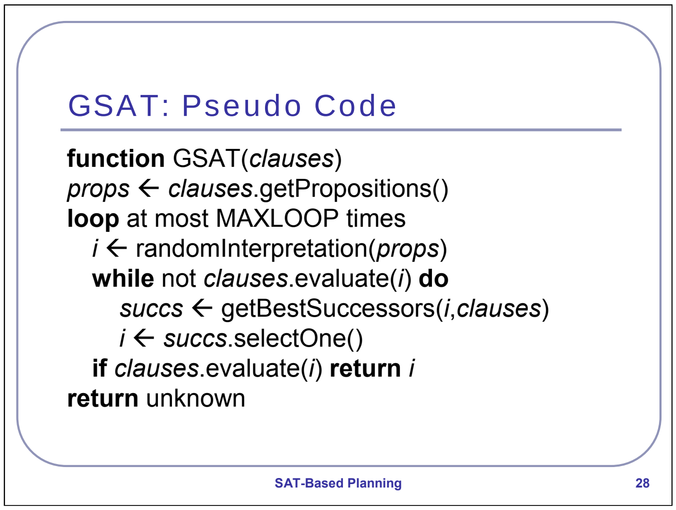
Random-Restart Hill-Climbing

- **method:**
  - must be able to generate random states; therefore usually complete state formulation
  - **conduct a series of hill-climbing searches from randomly generated initial states**
  - **stop when a goal is found**
- **analysis:**
  - **complete with probability approaching 1**
  - **requires 1/*p* restarts where *p* is the probability of success (1 success + 1/*p*-1 failures)**
  - no guarantee that algorithm will terminate; hence no time complexity
  - space complexity: *b*, the branching factor
  - completeness: eventually, it will generate the global maximum
  - optimality: same as completeness



Hill Climbing: getBestSuccessors

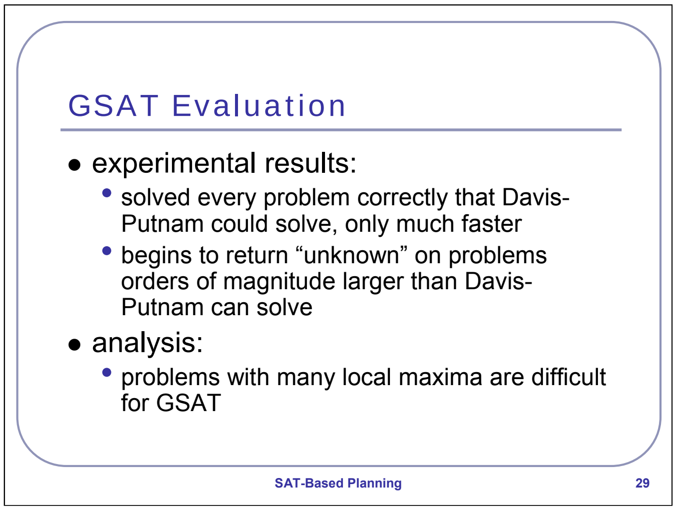
- **getBestSuccessors(*i*,*clauses*)**
  - given an interpretation and a set of clauses, find best "neighbours"
- ***tc* ← -1; *succs* ← {}**
  - *tc*: number of true clauses under best neighbour interpretation found so far
- **for every proposition *p* in *i***
- ***i'* ← *i*.flipValueOf(*p*)**
  - copy of given interpretation with value of *p* flipped
- ***n* ← number of *clauses* true under *i'***
- **if *n* > *tc* then *tc* ← *n*; *succs* ← {}**
  - if this interpretation is better then previous interpretations, remember how good it is and forget previous successors
- **if *n* = *tc* then *succs* ← *succs* + *i'***
  - if this interpretation is as good as the best so far, add it to the set of successors
- **return *succs***



GSAT: Pseudo Code

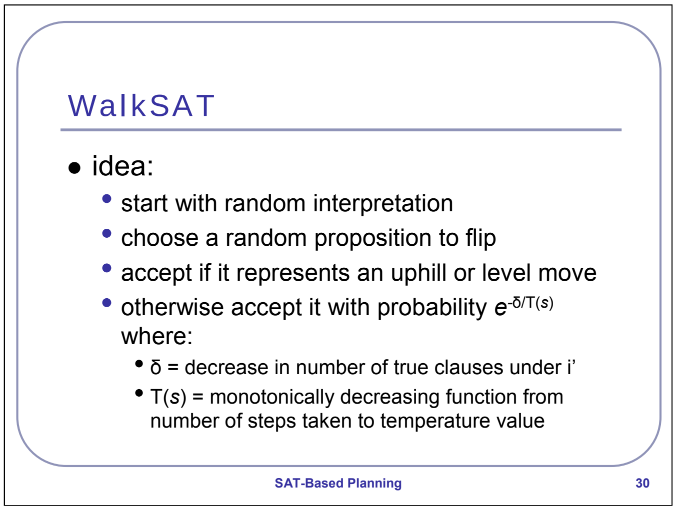
- **function GSAT(*clauses*)**
  - returns a model or failure
- ***props* ← *clauses*.getPropositions()**
- **loop at most MAXLOOP times**
  - ensures that the algorithm terminates
- ***i* ← randomInterpretation(*props*)**
- **while not *clauses*.evaluate(*i*) do**
  - while the interpretation is not a model
- ***succs* ← getBestSuccessors(*i*,*clauses*)**
- ***i* ← *succs*.selectOne()**
  - random selection, no backtracking
- **if *clauses*.evaluate(*i*) return *I***
  - found a model: clauses are satisfiable, return model
- **return unknown**
  - there may be a model, but we didn’t find one

- variants:
  - break inner loop when no improvement is made
  - allow only a limited number of level steps



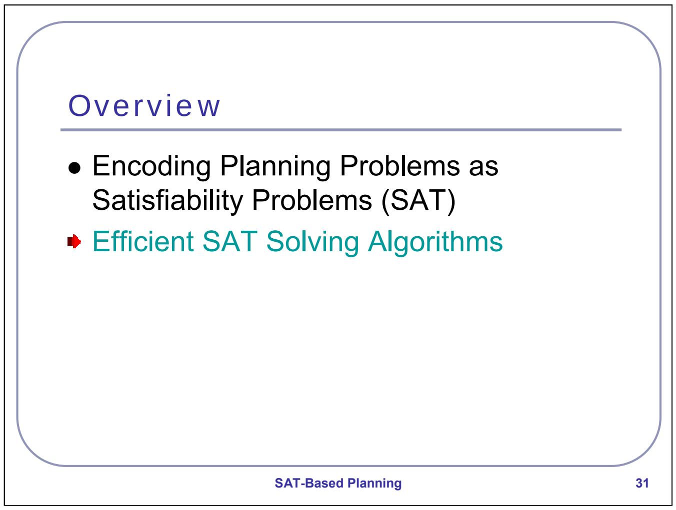
GSAT Evaluation

- **experimental results:**
    - **solved every problem correctly that Davis-Putnam could solve, only much faster**
    - **begins to return “unknown” on problems orders of magnitude larger than Davis-Putnam can solve**
- **analysis:**
    - **problems with many local maxima are difficult for GSAT**



WalkSAT

- inspired by simulated annealing
- **idea:**
  - **start with random interpretation**
  - **choose a random proposition to flip**
  - **accept if it represents an uphill or level move**
  - **otherwise accept it with probability $e^{-\delta/T(s)}$ where:**
    - **δ = decrease in number of true clauses under i’**
    - **T(*s*) = monotonically decreasing function from number of steps taken to temperature value**



Overview

Encoding Planning Problems as Satisfiability Problems (SAT)

- **Efficient SAT Solving Algorithms**
  - just done: efficient algorithms for SAT solving (very quick overview)