Lecture 7b: SAT Based Planning

CSCI 360

Introduction to Artificial Intelligence

USC

Here is where we are…

Week	30000D	30282R	Topics	Chapters
1	1/7	1/8	Intelligent Agents	[Ch 1.1-1.4 and 2.1-2.4]
	1/9	1/10	Problem Solving and Search	[Ch 3.1-3.3]
2	1/14	1/15	Uninformed Search	[Ch 3.3-3.4]
	1/16	1/17	Heuristic Search (A*)	[Ch 3.5]
3	1/21	1/22	Heuristic Functions	[Ch 3.6]
	1/23	1/24	Local Search	[Ch 4.1-4.2]
	1/25		Project 1 Out	
4	1/28	1/29	Adversarial Search	[Ch 5.1-5.3]
	1/30	1/31	Knowledge Based Agents	[Ch 7.1-7.3]
5	2/4	2/5	Propositional Logic Inference	[Ch 7.4-7.5]
	2/6	2/7	First-Order Logic	[Ch 8.1-8.4]
	2/8		Project 1 Due	
	2/8		Homework 1 Out	
6	2/11	2/12	Rule-Based Systems	[Ch 9.3-9.4]
	2/13	2/14	Search-Based Planning	[Ch 10.1-10.3]
	2/15		Homework 1 Due	
7	2/18	2/19	SAT-Based Planning	[Ch 10.4]
	2/20	2/21	Knowledge Representation	[Ch 12.1-12.5]
8	2/25	2/26	Midterm Review	
	2/27	2/28	**Midterm Exam**	

Outline

- What is AI?
- Problem-solving agent
  - Uninformed (DFS), informed (A*), and local search
  - Adversarial search (minimax, alpha-beta pruning)
- **Knowledge-based agent**
  - Propositional Logic
  - First Order Logic (FOL)
  - Graph-based Planning
  - **SAT-based Planning**
    - Modern SAT Solvers
    - SAT-based Planning

Recap: *Difference between "search" and "planning"*

- **Problem-solving agent** can find a sequence of actions that result in a goal state
  - it deals with "**atomic"** representations of states
  - Needs **"domain-specific"** heuristics to perform well in search

- **Planning agent** can also find a sequence of actions that result in a goal state
  - But it uses a **"factored"** representations of states
  - Can have **"generic"** heuristics for search

Logic formulas in a restricted format

Recap: *Search vs. planning*

- Planning opens up **action** and **goal** representations

	Search	**Planning**
States		
Actions		
Goal		
Plan		

Recap: *Search vs. planning*

- Planning opens up **action** and **goal** representations

	Search	**Planning**
States	data structures	Logical sentences
Actions	code	Preconditions/outcomes
Goal	code	Logical sentence (conjunction)
Plan	Sequence from S_0	Constraints on actions

Recap: *Planning Domain Definition Language*

- It uses a restricted subset of first-order logic (FOL) to make planning **efficiently solvable**

- **State:** a conjunction of functionless ground literals

$$Poor \wedge Unknown$$

$$At(Truck_1, Melbourne) \wedge At(Truck_2, Sydney)$$

$At(x, y)$ cannot have variables (x,y)

$At(Father(Fred), Sydney)$ cannot have function symbol

- **Goal:** a conjunction of literals, but may have variables

$$At(Home) \wedge Have(Milk) \wedge Have(Bananas) \wedge Have(Drill)$$

$$At(x) \wedge Sells(x, Milk)$$

Recap: *Planning Domain Definition Language*

- It uses a restricted subset of first-order logic (FOL) to make planning **efficiently solvable**
- **State:** a conjunction of functionless ground literals
- **Actions:**

Action name

$$\begin{array}{l} Action(Fly(p, from, to), \\ \quad \text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) \\ \quad \text{EFFECT:} \neg At(p, from) \wedge At(p, to)) \end{array}$$

Conjunction of ***positive*** literals

Conjunction of literals ***(positive or negative)***

Recap: *Planning Domain Definition Language*

- It uses a restricted subset of first-order logic (FOL) to make planning **efficiently solvable**

- **State:** a conjunction of functionless ground literals

- **Actions:**

$$
\begin{aligned}
&Action(Fly(p, from, to),\\
&\quad \text{PRECOND:} At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad \text{EFFECT:} \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Negative literal

DEL this lieteral
from the new state

Positive literal

ADD this lieteral
into the new state

Recap: *Planning Domain Definition Language*

- It uses a restricted subset of first-order logic (FOL) to make planning **efficiently solvable**
- **State:** a conjunction of functionless ground literals
- **Actions:**

$$\begin{array}{l} Action(Fly(p, from, to), \\ \quad \text{PRECOND:} At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to) \\ \quad \text{EFFECT:} \neg At(p, from) \wedge At(p, to)) \end{array}$$

- **Transition model:**

$$\text{RESULT}(s, a) = (s - \text{DEL}(a)) \cup \text{ADD}(a)$$

Recap: *Example: Air cargo transportation*

Recap: *Example: Air cargo transportation*

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$

$Goal(At(C_1, JFK) \land At(C_2, SFO))$

$Action(Load(c, p, a),$
  PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
  EFFECT: $\neg At(c, a) \land In(c, p))$

$Action(Unload(c, p, a),$
  PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
  EFFECT: $At(c, a) \land \neg In(c, p))$

$Action(Fly(p, from, to),$
  PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
  EFFECT: $\neg At(p, from) \land At(p, to))$

Recap: *Example: Air cargo transportation*

$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK) \\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2) \\
&\quad \land Airport(JFK) \land Airport(SFO)) \\
&Goal(At(C_1, JFK) \land At(C_2, SFO)) \\
&Action(Load(c, p, a), \\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p)) \\
&Action(Unload(c, p, a), \\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a) \\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p)) \\
&Action(Fly(p, from, to), \\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to) \\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Recap: *Example: Air cargo transportation*

$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$
$\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$
$\quad \land Airport(JFK) \land Airport(SFO))$
$Goal(At(C_1, JFK) \land At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $\neg At(c, a) \land In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$
EFFECT: $At(c, a) \land \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$
EFFECT: $\neg At(p, from) \land At(p, to))$

Recap: *Example: Air cargo transportation*

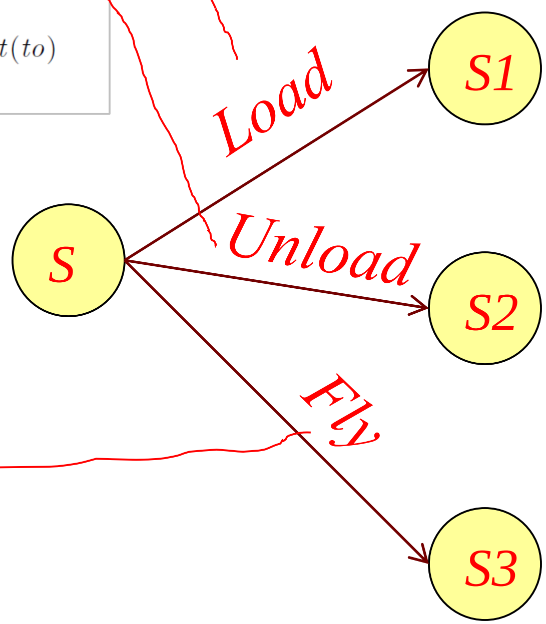
$$
\begin{aligned}
&Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)\\
&\quad \land Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)\\
&\quad \land Airport(JFK) \land Airport(SFO))\\
&Goal(At(C_1, JFK) \land At(C_2, SFO))\\
&Action(Load(c, p, a),\\
&\quad \text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)\\
&\quad \text{EFFECT: } \neg At(c, a) \land In(c, p))\\
&Action(Unload(c, p, a),\\
&\quad \text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)\\
&\quad \text{EFFECT: } At(c, a) \land \neg In(c, p))\\
&Action(Fly(p, from, to),\\
&\quad \text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)\\
&\quad \text{EFFECT: } \neg At(p, from) \land At(p, to))
\end{aligned}
$$

Recap: *Example: Air cargo transportation*

$$Init(At(C_1, SFO) \land At(C_2, JFK) \land At(P_1, SFO) \land At(P_2, JFK)$$
$$\land\ Cargo(C_1) \land Cargo(C_2) \land Plane(P_1) \land Plane(P_2)$$
$$\land\ Airport(JFK) \land Airport(SFO))$$
$$Goal(At(C_1, JFK) \land At(C_2, SFO))$$
$$Action(Load(c, p, a),$$
$$\text{PRECOND: } At(c, a) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT: } \neg At(c, a) \land In(c, p))$$
$$Action(Unload(c, p, a),$$
$$\text{PRECOND: } In(c, p) \land At(p, a) \land Cargo(c) \land Plane(p) \land Airport(a)$$
$$\text{EFFECT: } At(c, a) \land \neg In(c, p))$$
$$Action(Fly(p, from, to),$$
$$\text{PRECOND: } At(p, from) \land Plane(p) \land Airport(from) \land Airport(to)$$
$$\text{EFFECT: } \neg At(p, from) \land At(p, to))$$

Recap: *Example: Air cargo transportation*

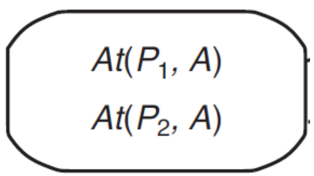
$Init(At(C_1, SFO) \wedge At(C_2, JFK) \wedge At(P_1, SFO) \wedge At(P_2, JFK)$
$\quad \wedge\ Cargo(C_1) \wedge Cargo(C_2) \wedge Plane(P_1) \wedge Plane(P_2)$
$\quad \wedge\ Airport(JFK) \wedge Airport(SFO))$
$Goal(At(C_1, JFK) \wedge At(C_2, SFO))$
$Action(Load(c, p, a),$
PRECOND: $At(c, a) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $\neg At(c, a) \wedge In(c, p))$
$Action(Unload(c, p, a),$
PRECOND: $In(c, p) \wedge At(p, a) \wedge Cargo(c) \wedge Plane(p) \wedge Airport(a)$
EFFECT: $At(c, a) \wedge \neg In(c, p))$
$Action(Fly(p, from, to),$
PRECOND: $At(p, from) \wedge Plane(p) \wedge Airport(from) \wedge Airport(to)$
EFFECT: $\neg At(p, from) \wedge At(p, to))$



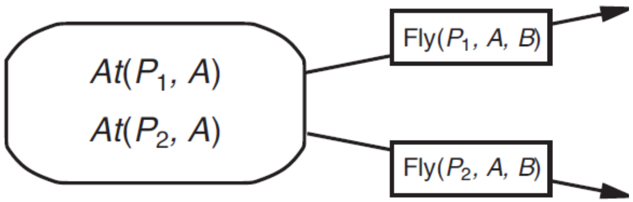
The following plan is a solution to the problem:

$$[Load(C_1, P_1, SFO), Fly(P_1, SFO, JFK), Unload(C_1, P_1, JFK),$$
$$Load(C_2, P_2, JFK), Fly(P_2, JFK, SFO), Unload(C_2, P_2, SFO)]\ .$$

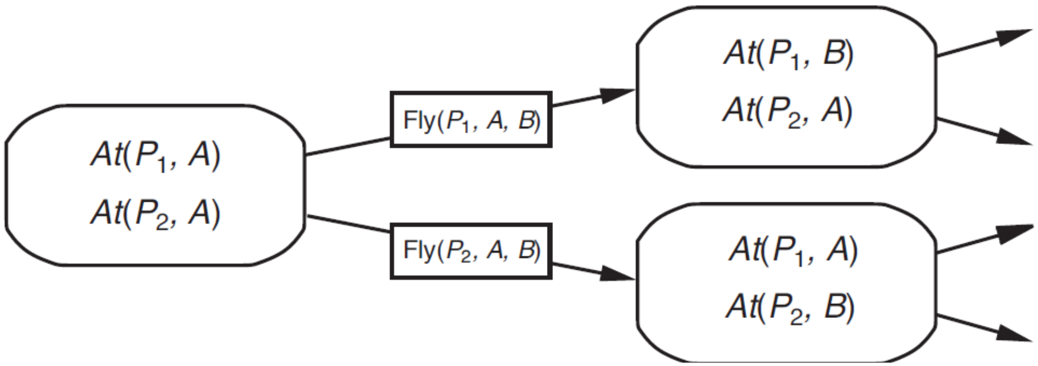
Recap: *Planning as state-space search* *(forward)*



Recap: *Planning as state-space search* *(forward)*



Recap: *Planning as state-space search* *(forward)*



Recap: *Heuristics for planning*

- Neither **forward** nor **backward** search is efficient without a good ***heuristic function***
  - Need an admissible heuristic
  - i.e., never overestimate the distance from state (s) to the goal

Question: Instead of designing a different heuristic function <u>for each and every planning problem</u> that you may encounter, can you design a "generic" heuristic that <u>works for all planning problems</u>?

Recap: *Planning graph*

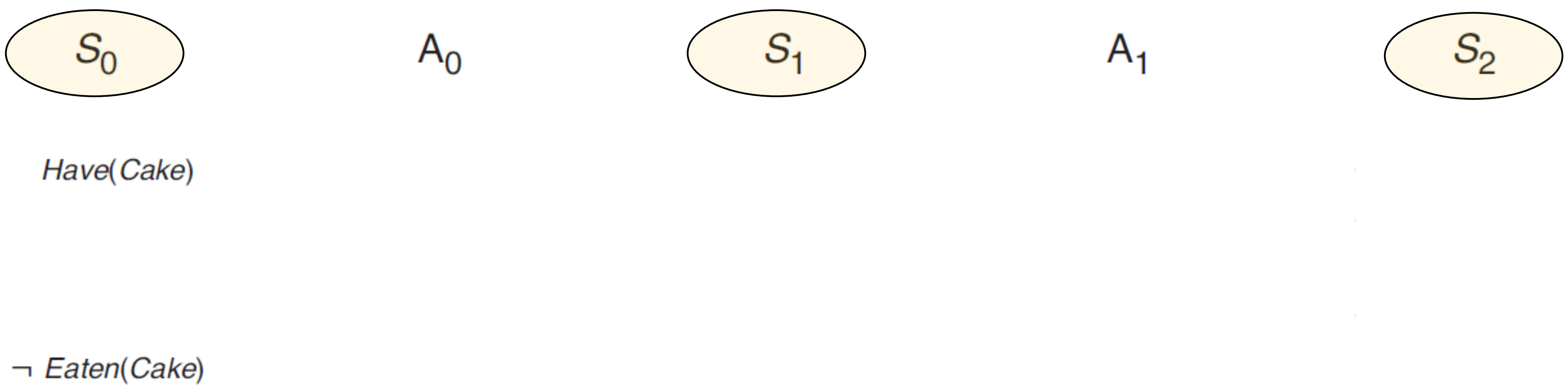
- It is a generic **data structure** that can be used to give an admissible **heuristic estimate** for any planning problem
  - Will never overestimate; and often accurate

$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$

Recap: *Planning graph*

- S0, S1, S2 – states
  - Literals may be reached at each time step
  - Mutual exclusion links
- A0, A1 – actions
  - Mutual exclusion links

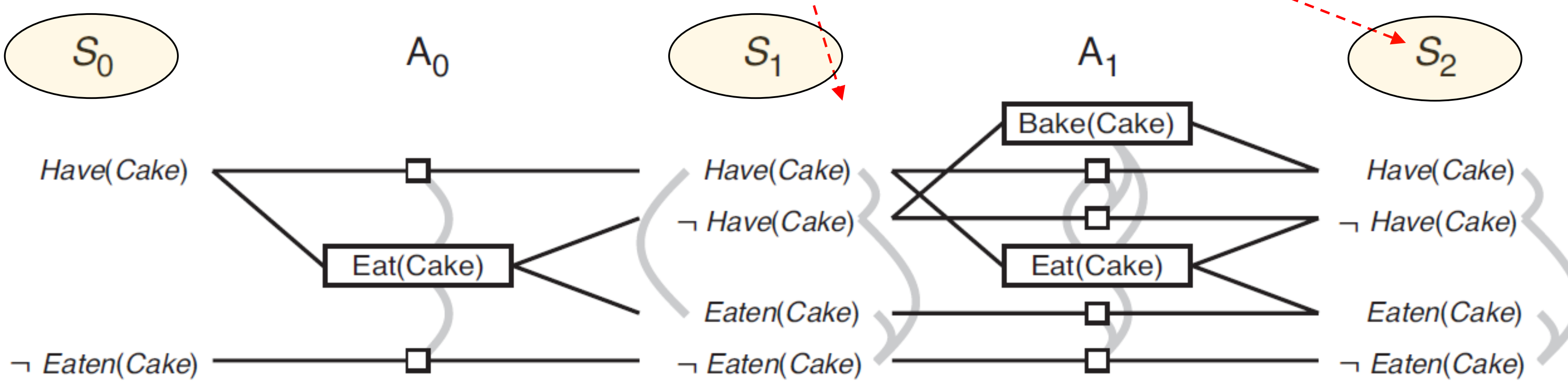
$Init(Have(Cake))$
$Goal(Have(Cake) \wedge Eaten(Cake))$
$Action(Eat(Cake)$
  PRECOND: $Have(Cake)$
  EFFECT: $\neg Have(Cake) \wedge Eaten(Cake))$
$Action(Bake(Cake)$
  PRECOND: $\neg Have(Cake)$
  EFFECT: $Have(Cake))$



Recap: *Planning graph*

- S0, S1, S2 – states
  - Literals may be reachable at each level
  - Mutual exclusion (mutex) links
- A0, A1 – actions
  - – Mutual exclusion (mutex) links

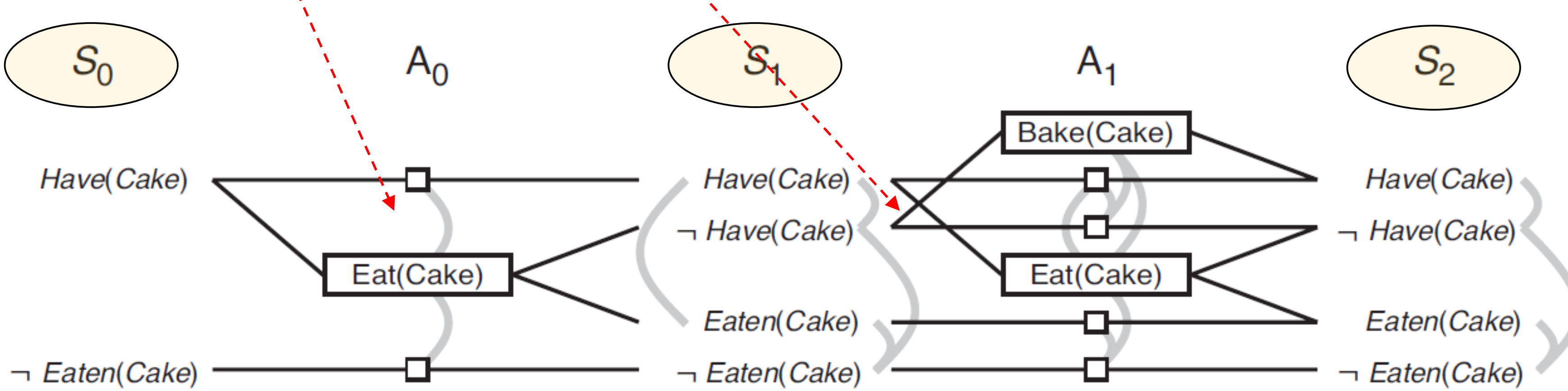
Init(*Have*(*Cake*))
Goal(*Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Eat*(*Cake*)
  PRECOND: *Have*(*Cake*)
  EFFECT: ¬ *Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Bake*(*Cake*)
  PRECOND: ¬ *Have*(*Cake*)
  EFFECT: *Have*(*Cake*))



Planning graph: *mutex actions*

- Actions: Effects contradict each other
  - Eat(Cake).effect vs. Have(Cake).effect
- Actions: Preconditions contradict each other
  - Bake(Cake).precond vs Eat(Cake).precond

Init(*Have*(*Cake*))
Goal(*Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Eat*(*Cake*)
  PRECOND: *Have*(*Cake*)
  EFFECT: ¬ *Have*(*Cake*) ∧ *Eaten*(*Cake*))
Action(*Bake*(*Cake*)
  PRECOND: ¬ *Have*(*Cake*)
  EFFECT: *Have*(*Cake*))

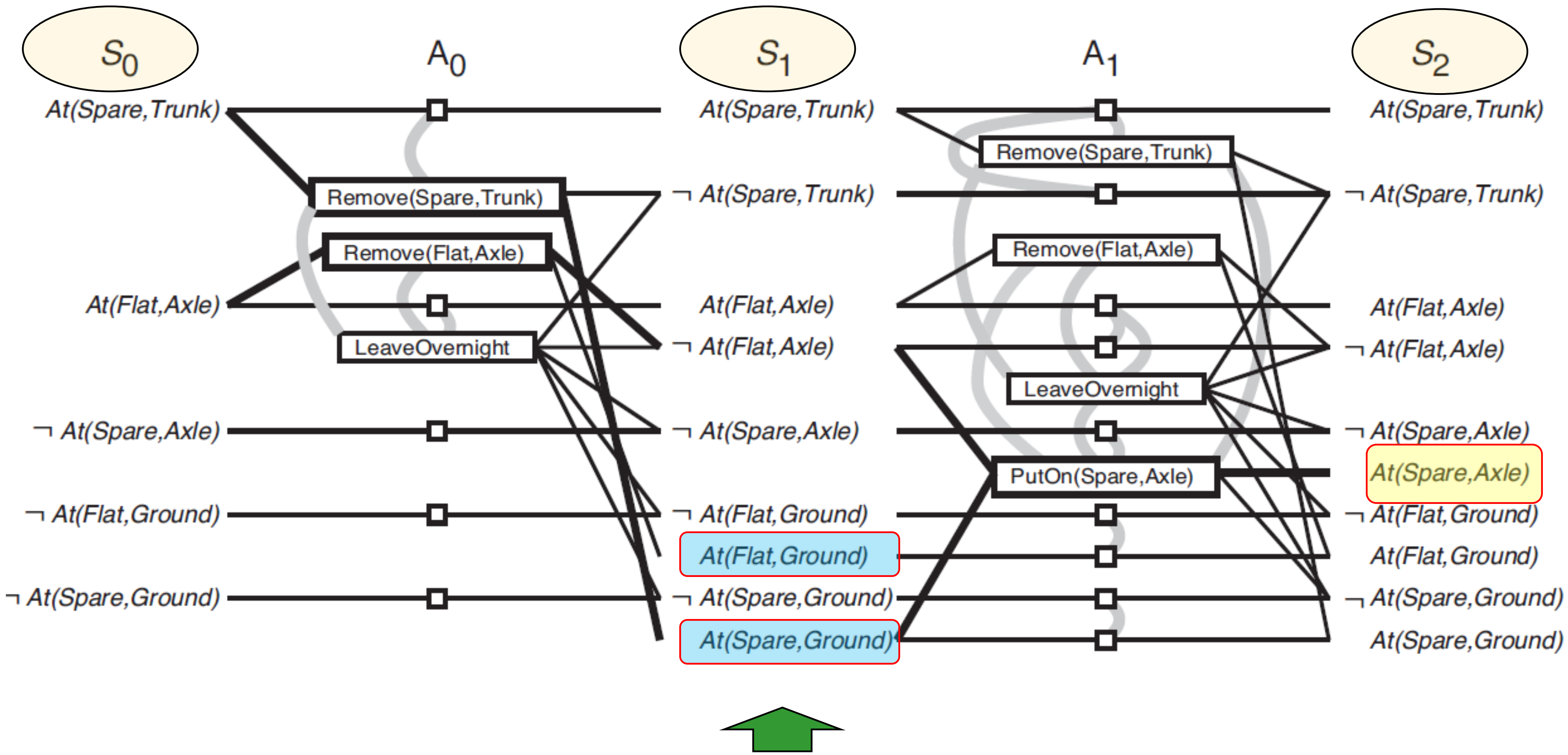


Properties of a planning graph

- **Polynomial** in the size of the planning problem
  - Instead of being "**exponential**" in size 
- If any goal literal fails to appear in the final level of the graph, then the problem is **unsolvable**
  - Sound conclusion 
- The **cost** of achieving any **goal (*g*)** can be **estimated** as the level at which (***g***) first appears in the planning graph constructed from (***s***) as the initial state
  - Never over-estimate 

Example planning graph (spare tire)

Example planning graph (spare tire)



Reached more states that actually possible

Outline for today

- **Solving AI Planning Problems using SAT**
  - **What is SAT?**
  - **How to implement a SAT solver?**
  - **How to reduce an AI planning problem to SAT?**

History…

What happened in ***1996*** *and* ***2001****?*

- 1969 Plan synthesis as theorem proving (Green IJCAI-69)
- 1971 STRIPS (Fikes & Nilsson AIJ-71)
  - Decades of work on "specialized theorem provers"…
- 1992 SatPlan approach (Kautz & Selman ECAI-92)
  - Encoding STRIPS-style linear planning;
  - Didn't appear practical…
- **1996** (Kautz & Selman AAAI-96) (Kautz, McAllester, Selman KR-96)
  - Electrifying results (hand translated formulas)
- 1997 MEDIC (Ernst et al, IJCAI-97)
  - First complete implementation of SatPlan (with compiler)
- **2001** and beyond
  - Rapid progress on SAT solving techniques
- 2004 SatPlan was clear winner of "optimal planners"
  - Due to huge advances in SAT solvers (e.g., zChaff, miniSAT,…)

Satisfiability (SAT)

- SAT is the problem of determining if the variables of a Propositional logic (Boolean) formula can be assigned (to true/false) such that the formula evaluates to TRUE.

Theoretical Complexity

- The first **NP-complete** problem (Stephen Cook,1971)
  - All currently known algorithms are exponential-time
  - Not clear if a polynomial-time algorithm exists

- One of the seven **Millennium Prize Problems**
  - US$ 1,000,000 prize for the first correct solution

P versus NP problem:

Informally, it asks whether every problem whose solution can be quickly verified by a computer can also be quickly solved by a computer. It was introduced in 1971 by Stephen Cook.

Conjunctive Normal Form (re-cap)

- Boolean Variable
- Literal
- Clause
- CNF

Examples

$$(a \lor \underbrace{\neg c}_{\textit{literal}}) \land (b \lor \neg c) \land \underbrace{(\neg a \lor \neg b \lor c)}_{\textit{clause}} \land \ldots$$

Unit Propagation

- Remove “unit (one-literal) clauses”
  - Find them → set them to TRUE → remove clauses → propagate values → …

Examples

$$(b) \wedge (\neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg d)$$

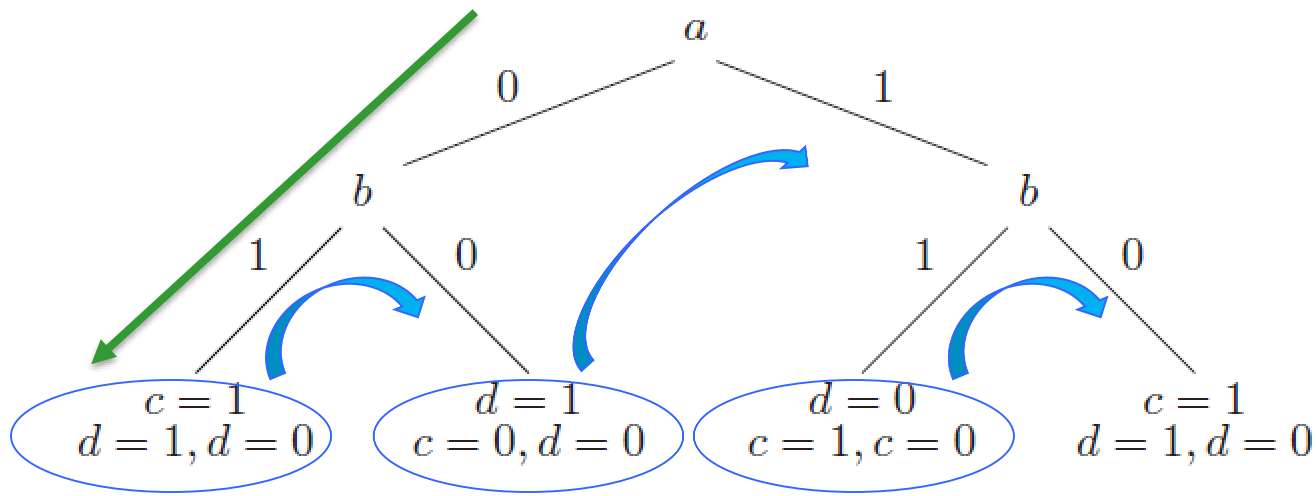
What is DPLL?

- A Classic SAT Algorithm
  - by Davis, Putnam, Logemann, Loveland [DP60, DLL62]

Making a Decision

- Pick a free variable and set it to either true or false

$$(\neg a \lor \neg b \lor c) \land (a \lor \neg b \lor c) \land (\neg c \lor d) \land (\neg c \lor \neg d) \land \\ (\neg a \lor c \lor d) \land (\neg a \lor b \lor \neg d) \land (b \lor c \lor \neg d) \land (a \lor b \lor d)$$



Deducing a Conflict → (Chronological) Backtracking

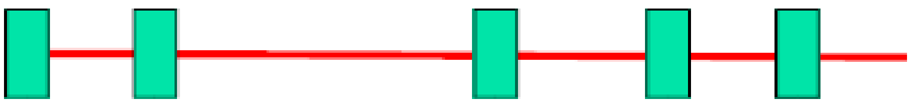
Exponential time complexity

Two Key Ideas in Modern SAT Solvers

- Conflict Driven Clause Learning (CDCL) [MS96]
- Fast Deduction and Backtracking [MMZ+01]

TimeLine

1960
DP
≈10 var



1962
DLL
≈ 10 var

1986
BDD
≈ 100 var

1992
GSAT
≈ 300 var

CDCL: The GRASP Story (1996)

GRASP—a new search algorithm for satisfiability

Full Text: Pdf Buy this Article

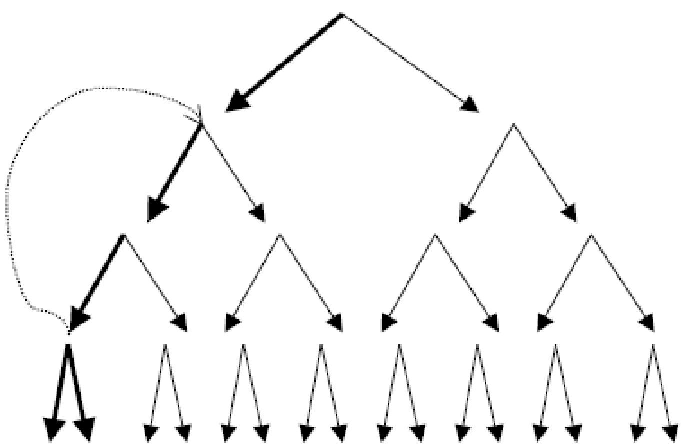
1997 Article

Authors: João P. Marques Silva Cadence European Laboratories IST/INESC, 1000 Lisboa, Portugal

Karem A. Sakallah Department of EECS, University of Michigan, Ann Arbor, Michigan

Bibliometrics

Search Tree

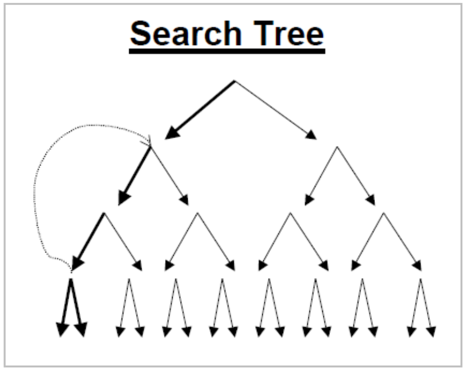


Backtracking

Chronological, or non-chronological

Decision Stack

- Main data structures of a SAT solver
  - Clause database
  - Assignment stack

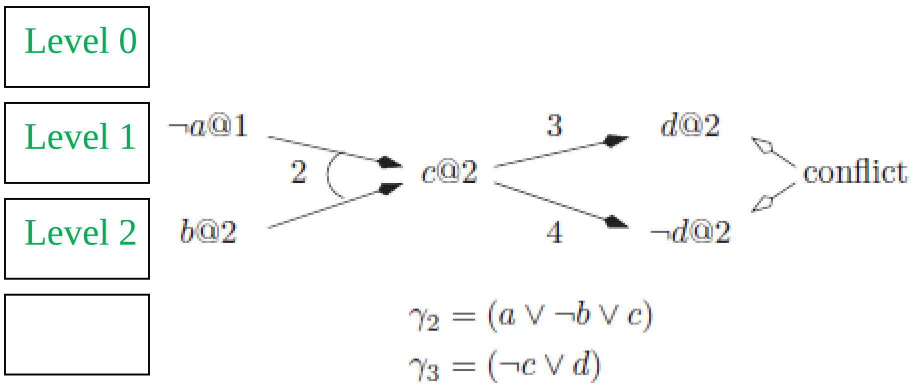


- Decision level
  - Pick a variable, set it to true or false, and derive all the implications
  - All these variables are assigned at the same “decision level”

Decision ➔ Implication ➔ Conflict

$$(\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$$

Decision stack

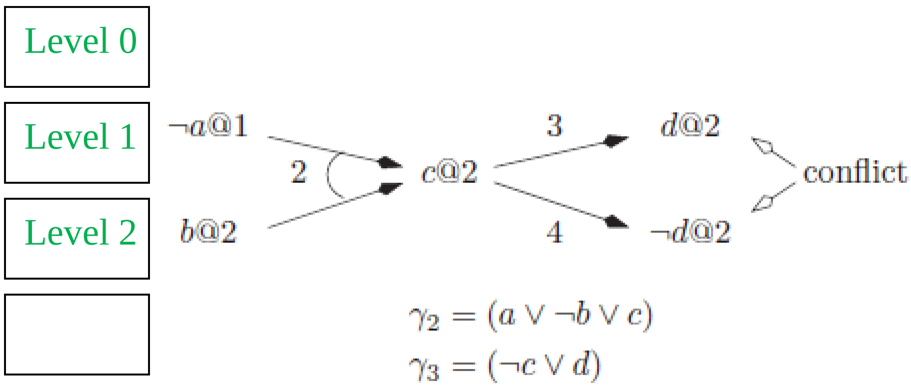


$$\gamma_2 = (a \vee \neg b \vee c)$$
$$\gamma_3 = (\neg c \vee d)$$
$$\gamma_4 = (\neg c \vee \neg d)$$

Conflict Analysis ➔ New (Learned) Clause

$(\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$

Decision stack



$\gamma_2 = (a \vee \neg b \vee c)$

$\gamma_3 = (\neg c \vee d)$

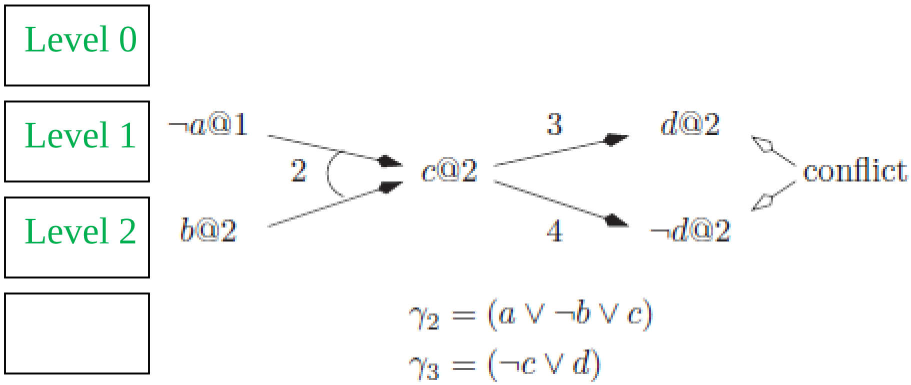
$\gamma_4 = (\neg c \vee \neg d)$

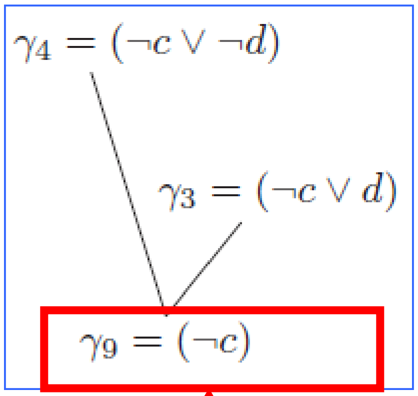
$\boxed{\gamma_9 = (\neg c)}$

New (Learned) Clause

$$(\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$$

Decision stack





$\gamma_2 = (a \vee \neg b \vee c)$

$\gamma_3 = (\neg c \vee d)$

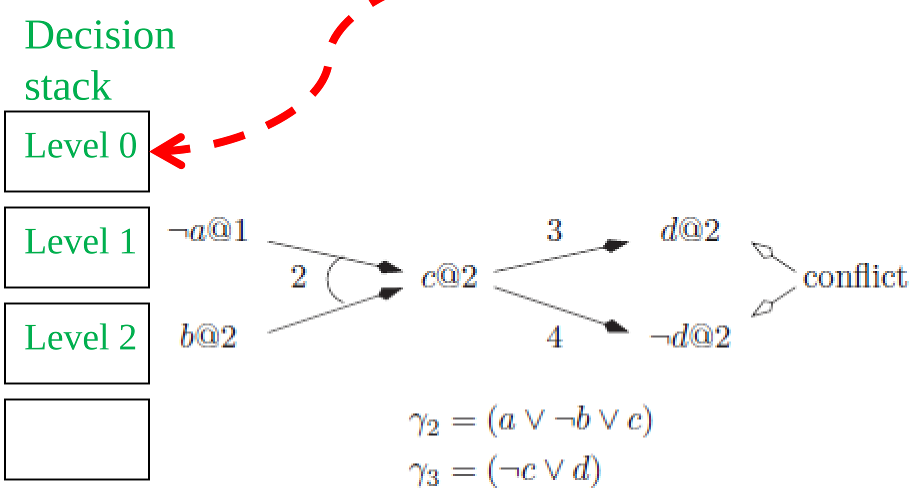
$\gamma_4 = (\neg c \vee \neg d)$

In theory, this clause may also be derived using "resolution" – it's a resolvent!

Non-chronological Backtracking

$(\neg a \vee b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d)$

$\wedge\ (\neg c)$



$\gamma_2 = (a \vee \neg b \vee c)$
$\gamma_3 = (\neg c \vee d)$
$\gamma_4 = (\neg c \vee \neg d)$

Augment the "clause database":

Without you making any decision, the newly added clause will trigger implications at Decision Level 0

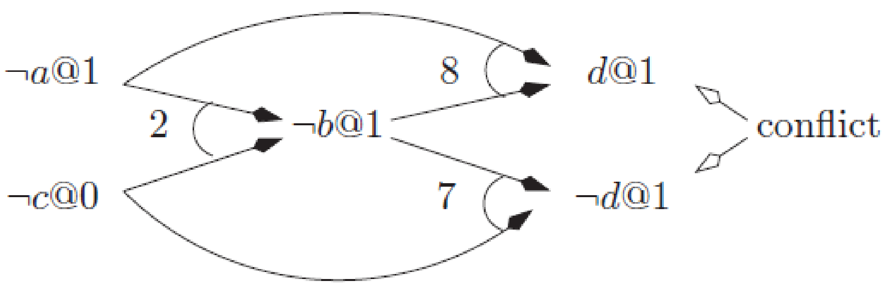
Conflict Analysis (larger example)

$$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$$

$$\wedge\ (\neg c)$$

Decision stack

Level 0
Level 1



$$\gamma_2 = (a \vee \neg b \vee c)$$

$$\gamma_7 = (b \vee c \vee \neg d)$$

$$\gamma_8 = (a \vee b \vee d)$$

$$\gamma_9 = (\neg c)$$

$$\gamma_{10} = (a \vee c)$$

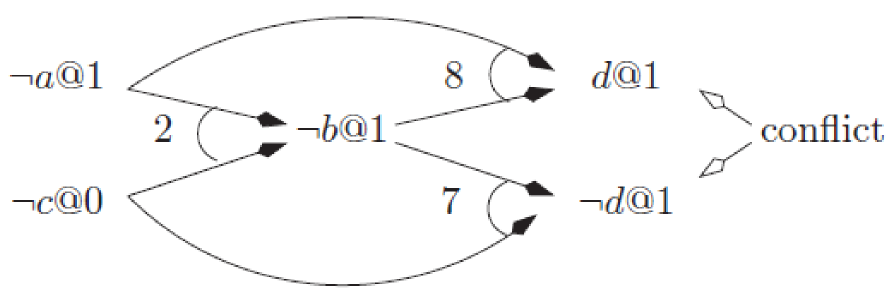
Conflict Analysis (larger example)

$$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$$

$\wedge\ (\neg c)$

Decision stack

Level 0
Level 1

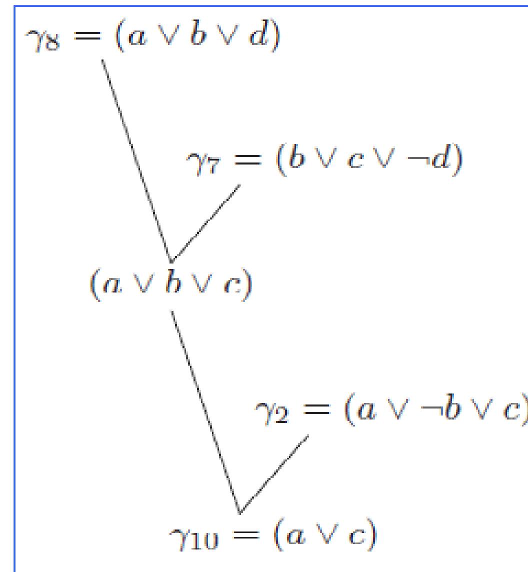


$\gamma_2 = (a \vee \neg b \vee c)$

$\gamma_7 = (b \vee c \vee \neg d)$

$\gamma_8 = (a \vee b \vee d)$

$\gamma_9 = (\neg c)$



In theory, this clause may also be derived using "resolution" – it's a resolvent!

Conflict Analysis (larger example)

$$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge$$
$$(\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$$

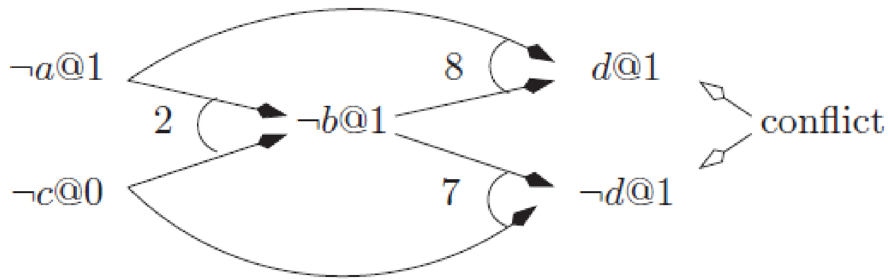
$\wedge\ (\neg c)$

$\wedge\ (a \vee c)$

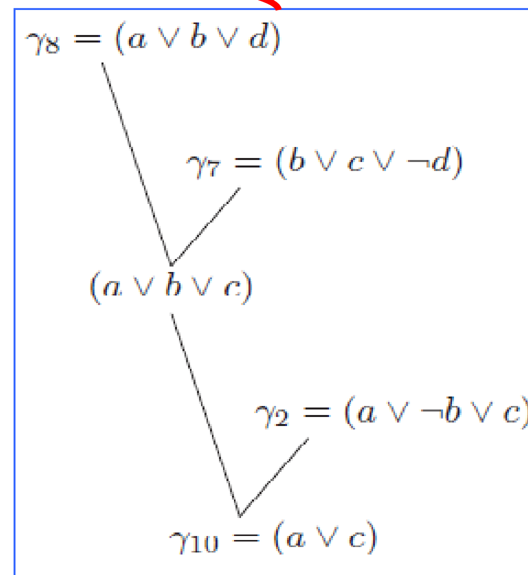
Decision stack

Level 0

Level 1



$\gamma_2 = (a \vee \neg b \vee c)$
$\gamma_7 = (b \vee c \vee \neg d)$
$\gamma_8 = (a \vee b \vee d)$
$\gamma_9 = (\neg c)$



Augment the “clause database”

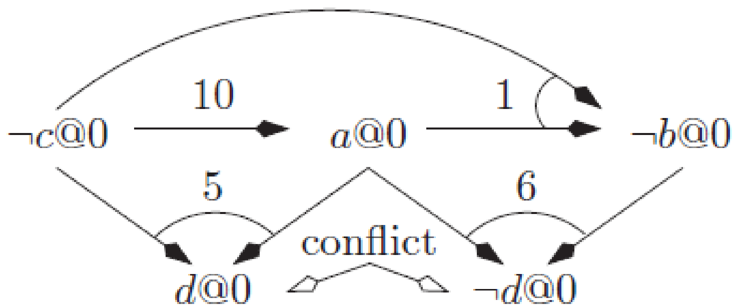
Conflict Analysis (larger example)

$$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$$ $\boldsymbol{\wedge (\neg c)}$

$\boldsymbol{\wedge (a \vee c)}$

Decision stack

Level 0
Level 1



$$\gamma_1 = (\neg a \vee \neg b \vee c)$$
$$\gamma_5 = (\neg a \vee c \vee d)$$
$$\gamma_6 = (\neg a \vee b \vee \neg d)$$
$$\gamma_9 = (\neg c)$$
$$\gamma_{10} = (a \vee c)$$

$$\gamma_{11} = (c)$$

Conflict Analysis (larger example)

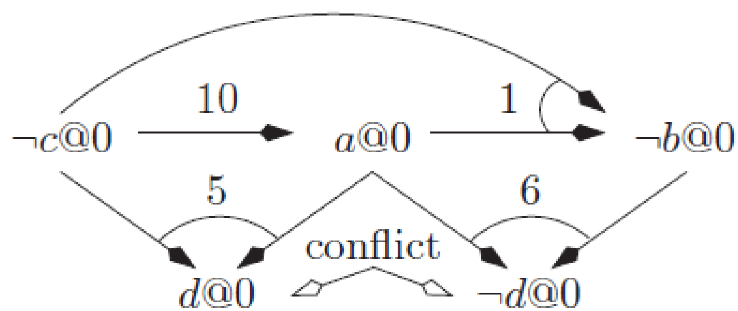
$$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge (\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$$

$\wedge\ (\neg c)$

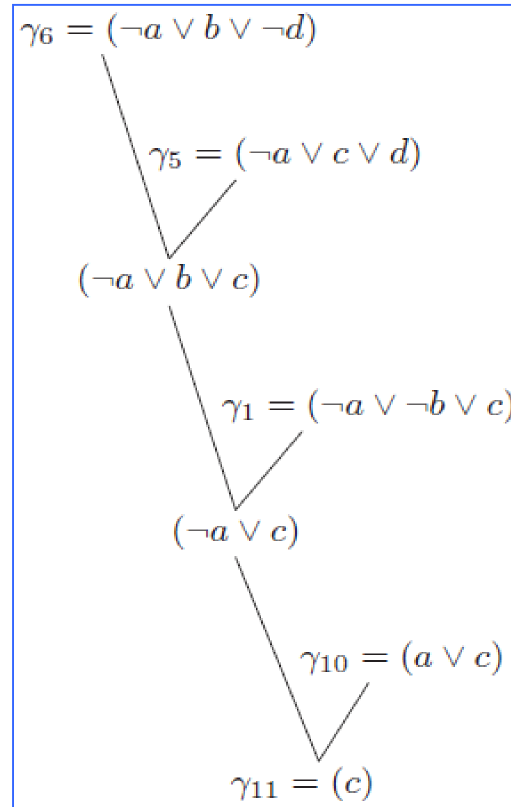
$\wedge\ (a \vee c)$

Decision stack

Level 0
Level 1



$$\gamma_1 = (\neg a \vee \neg b \vee c)$$
$$\gamma_5 = (\neg a \vee c \vee d)$$
$$\gamma_6 = (\neg a \vee b \vee \neg d)$$
$$\gamma_9 = (\neg c)$$
$$\gamma_{10} = (a \vee c)$$



In theory, this clause may also be derived using "resolution" – it's a resolvent!

Conflict Analysis (larger example)

$(\neg a \vee \neg b \vee c) \wedge (a \vee \neg b \vee c) \wedge (\neg c \vee d) \wedge (\neg c \vee \neg d) \wedge$
$(\neg a \vee c \vee d) \wedge (\neg a \vee b \vee \neg d) \wedge (b \vee c \vee \neg d) \wedge (a \vee b \vee d)$

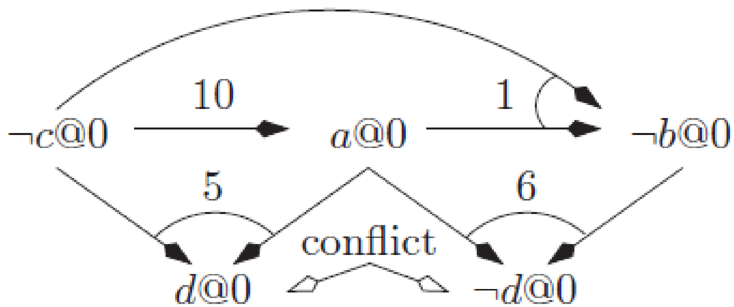
$\wedge\ (\neg c)$

$\wedge\ (a \vee c)$

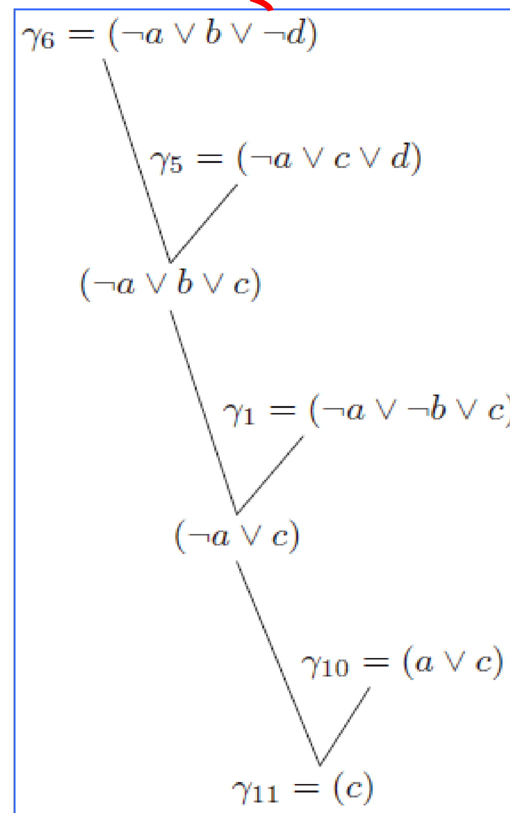
$\wedge\ (c)$

Decision stack

Level 0
Level 1



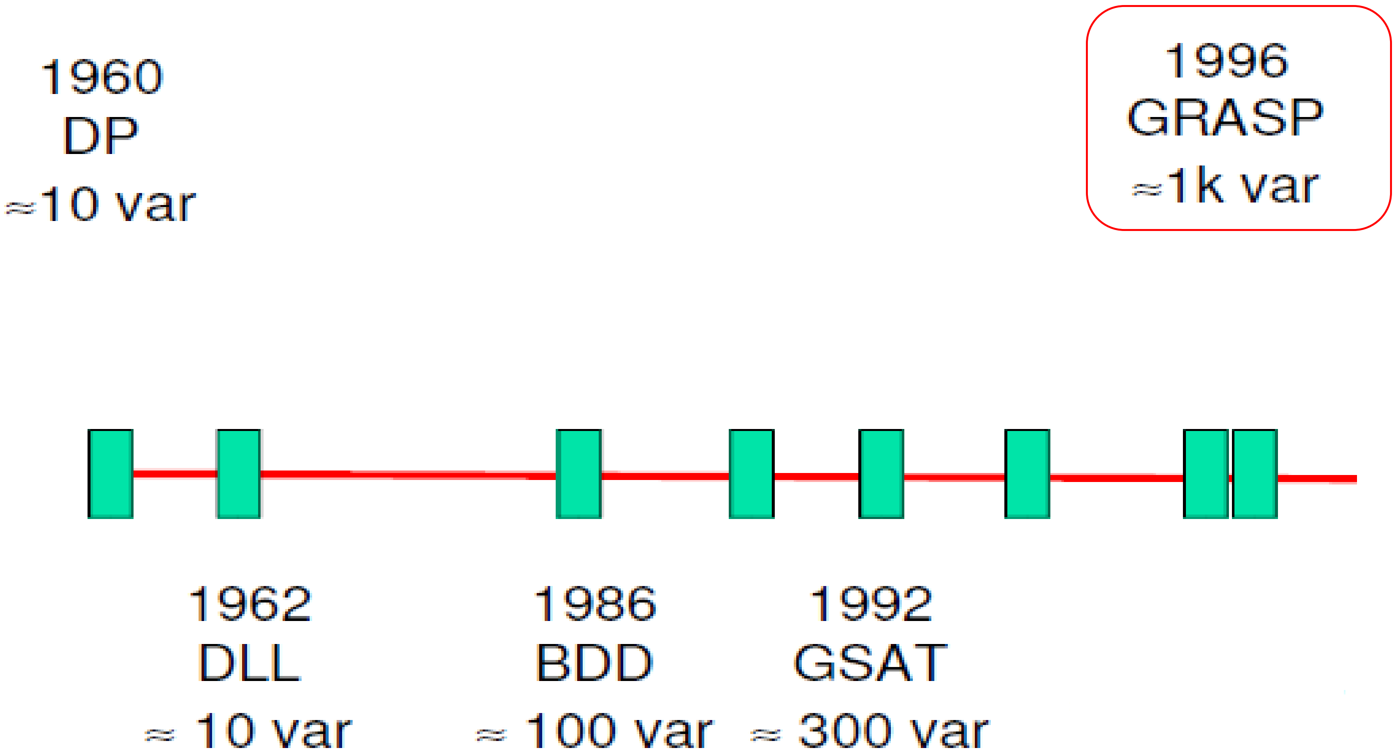
$$\gamma_1 = (\neg a \vee \neg b \vee c)$$
$$\gamma_5 = (\neg a \vee c \vee d)$$
$$\gamma_6 = (\neg a \vee b \vee \neg d)$$
$$\gamma_9 = (\neg c)$$
$$\gamma_{10} = (a \vee c)$$



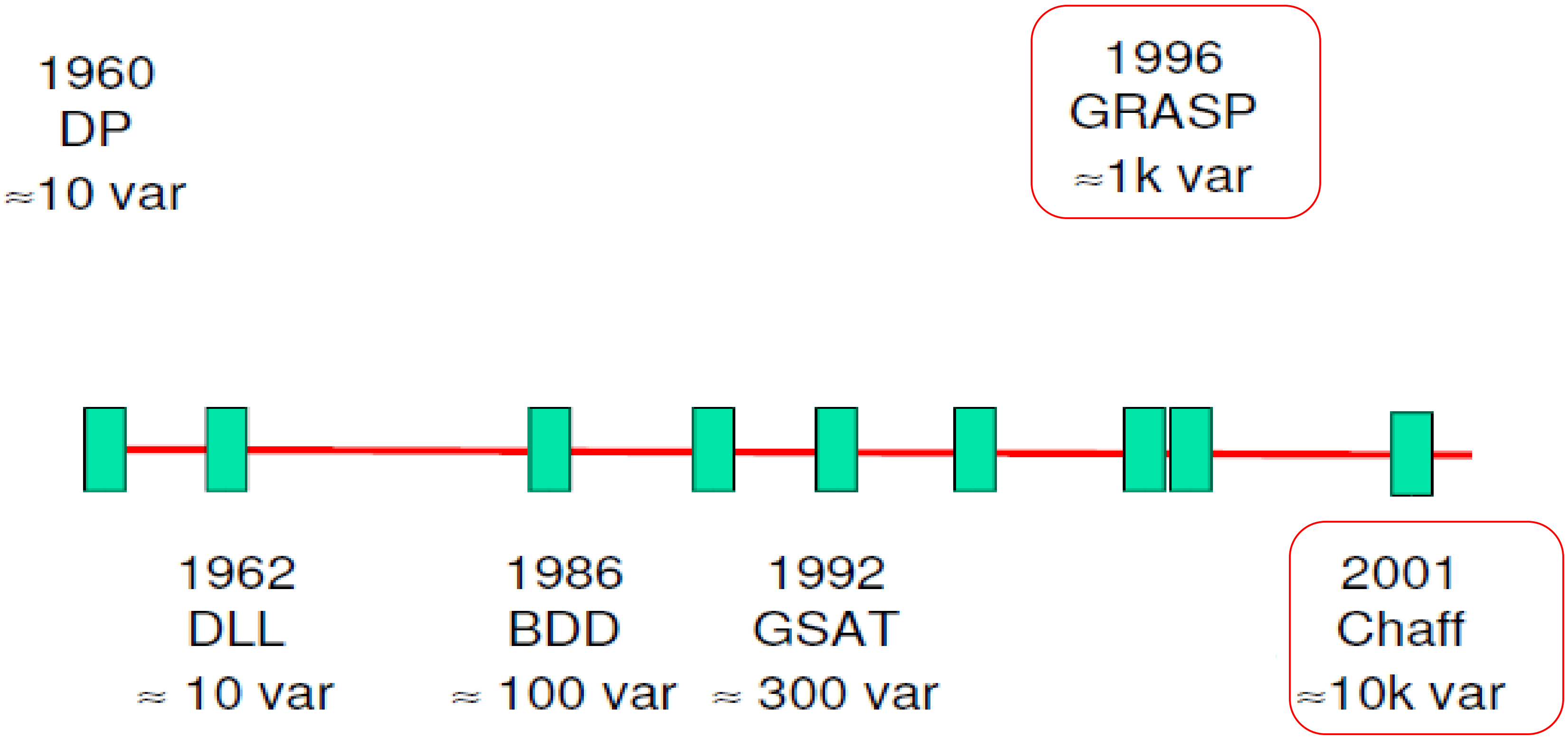
Impact of CDCL (conflict-driven clause learning)

- In practice, it has made many important SAT problems "**polynomial-time solvable**"

TimeLine


1960
DP
≈10 var
1996
GRASP
≈1k var
1962
DLL
≈ 10 var
1986
BDD
≈ 100 var
1992
GSAT
≈ 300 var

TimeLine


1960
DP
≈10 var
1996
GRASP
≈1k var
1962
DLL
≈ 10 var
1986
BDD
≈ 100 var
1992
GSAT
≈ 300 var
2001
Chaff
≈10k var

Key Ideas in Modern SAT Solvers

- Conflict Driven Clause Learning (CDCL) [MS96]
- Fast Deduction and Backtracking [MMZ+01]

The Chaff Story (Spring 2000)

- Conflict Driven Clause Learning (CDCL) [MS96]
- **Fast Deduction [MMZ+01]**

Prof. Shard Malik
Princeton University

M. Moskewicz
Student (senior)

C. Madigan
Student (senior)

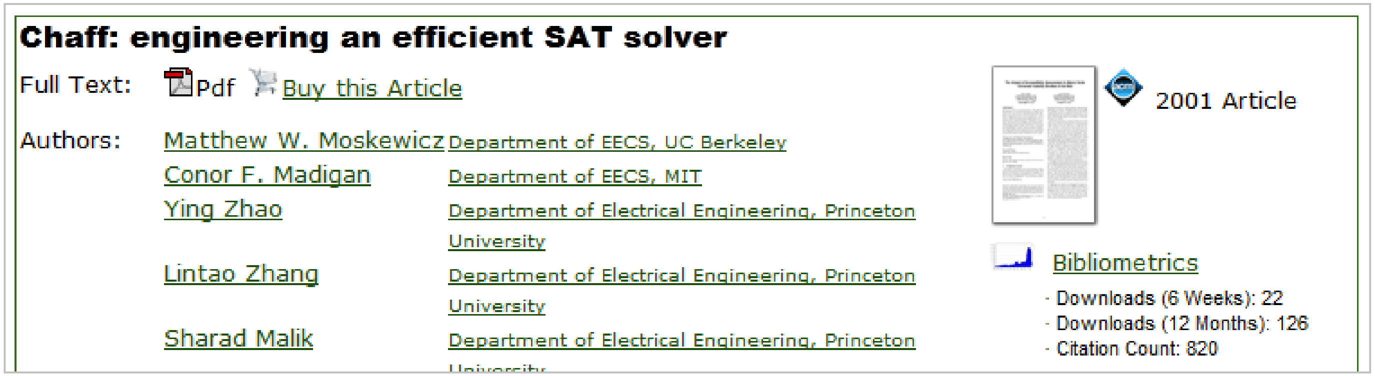
Class of 2000

Y. Zhao
Grad. student

L. Zhang
Grad. student

http://www.princeton.edu/pr/pwb/01/0305/1b.shtml

The Chaff Story (2009)



Chaff: engineering an efficient SAT solver

Full Text: Pdf Buy this Article

2001 Article

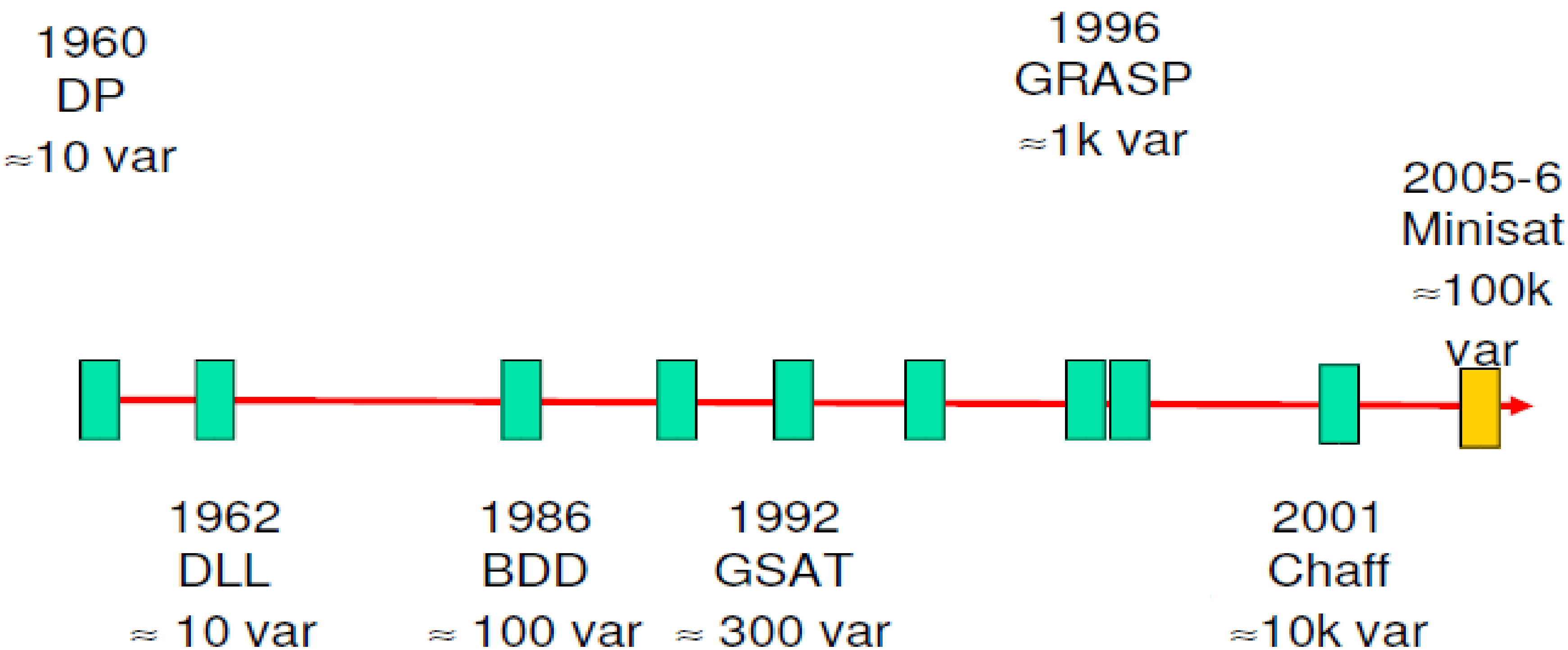
Authors:
- Matthew W. Moskewicz — Department of EECS, UC Berkeley
- Conor F. Madigan — Department of EECS, MIT
- Ying Zhao — Department of Electrical Engineering, Princeton University
- Lintao Zhang — Department of Electrical Engineering, Princeton University
- Sharad Malik — Department of Electrical Engineering, Princeton University

Bibliometrics
- Downloads (6 Weeks): 22
- Downloads (12 Months): 126
- Citation Count: 820

CAV Award 2009 ($10,000)

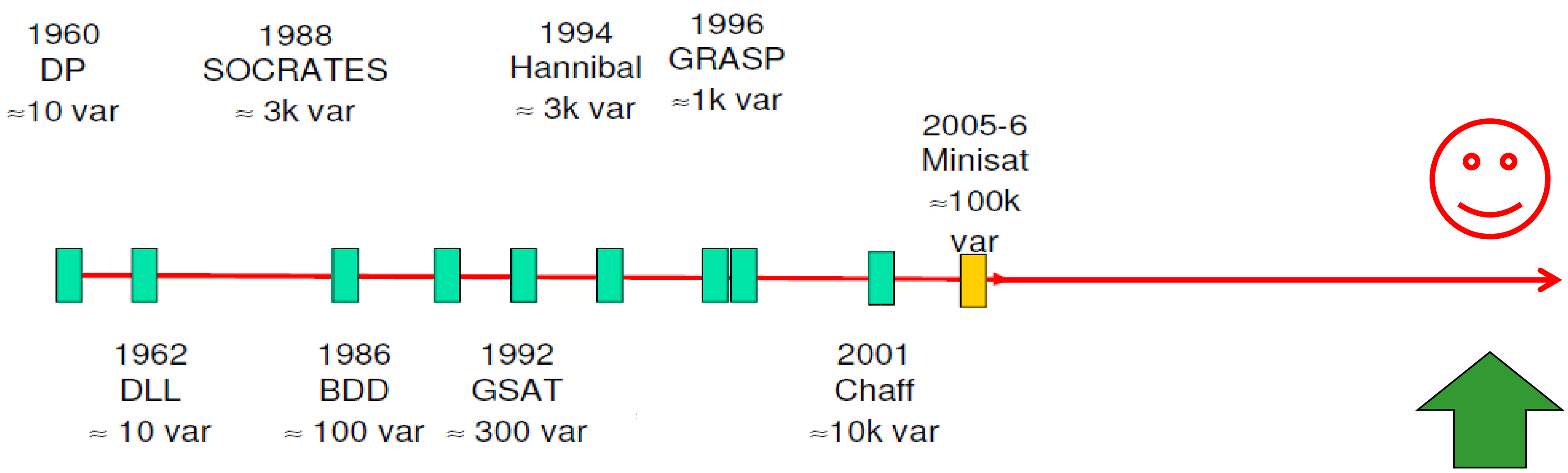
http://www.prlog.org/10326322-michigan-and-princeton-researchers-recognized-for-work-on-boolean-satisfiability-sat-solvers.html

TimeLine



1960
DP
≈10 var
1996
GRASP
≈1k var
2005-6
Minisat
≈100k
var
1962
DLL
≈ 10 var
1986
BDD
≈ 100 var
1992
GSAT
≈ 300 var
2001
Chaff
≈10k var

What's Next? … Parallel SAT???



Some Applications of SAT Solvers

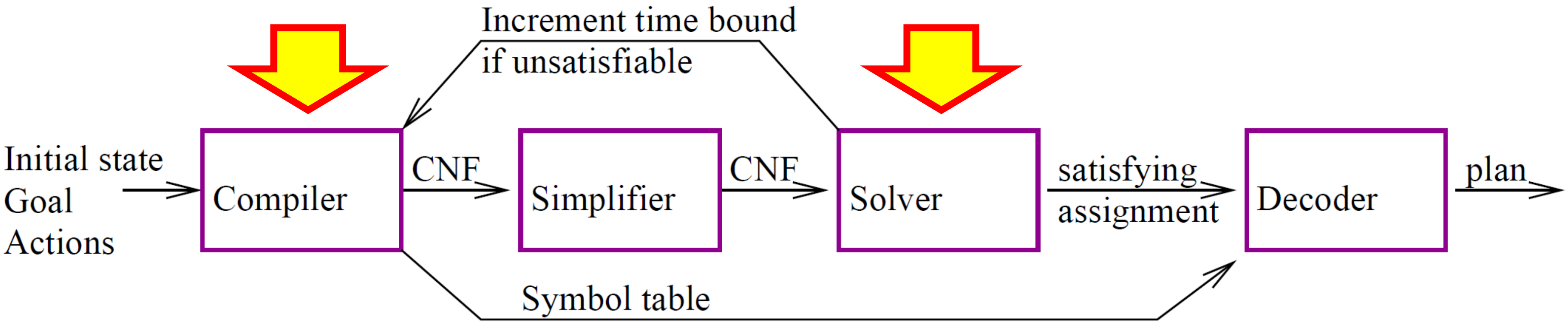
- Formal methods :
  - Hardware model checking ; Software model checking ; Termination analysis of term-rewrite systems ; Test pattern generation (testing of software & hardware) ; etc.
- Artificial intelligence :
  - Planning ; Knowledge representation ; Games (n-queens, sudoku, social golpher's, etc.)
- Bioinformatics :
  - Haplotype inference ; Pedigree checking ; Analysis of Genetic Regulatory Networks ; etc.
- Design automation :
  - Equivalence checking ; Delay computation ; Fault diagnosis ; Noise analysis ; etc.
- Security :
  - Cryptanalysis ; Inversion attacks on hash functions ; etc.

Outline for today

- **Solving AI Planning Problem using SAT**
  - **What is SAT?**
  - **How to implement a SAT solver?**
  - **How to reduce AI planning to SAT?**

High-level Procedure

- Initial state: where do we start?
- Goal state: where do we want to go?
- Transition: which action at time 0, time 1, and so on



SAT encoding of the planning problem

- Construct a propositional formula Φ for (P,n) such that
  - The propositional formula Φ is satisfiable if and only if there exists a plan $\pi = (a_0, a_1, \ldots, a_{n-1})$ for (P,n)
  - Every model of the formula Φ corresponds to a solution plan π

Constructing the CNF formulas

- Initial state
  - True at time 0 $\bigwedge\{l_0 \mid l \in s_0\} \wedge \bigwedge\{\neg l_0 \mid l \in L - s_0\}$
- Goal state
  - True at time n $\bigwedge\{l_n \mid l \in g^+\} \wedge \bigwedge\{\neg l_n \mid l \in g^-\}$

Constructing the CNF formulas

- Initial state
  - True at time 0 $\bigwedge\{l_0 \mid l \in s_0\} \wedge \bigwedge\{\neg l_0 \mid l \in L - s_0\}$
- Goal state
  - True at time n $\bigwedge\{l_n \mid l \in g^+\} \wedge \bigwedge\{\neg l_n \mid l \in g^-\}$
- Transition model
  - Actions and effects at time 0, time 1, and so on

    $a_i \Rightarrow \bigwedge\{p_i \mid p \in \text{Precond}(a)\} \wedge \bigwedge\{e_{i+1} \mid e \in \text{Effects}(a)\}$
  - Complete exclusion axioms
  - Frame axioms

- Formula Φ is the conjunction of these formulas
  - The solution plan $\pi = (a_0, a_1, \ldots, a_{n-1})$ is a sequence of actions

Complete exclusion axiom

- There can be "one and only one" action at each time step
  - Any two actions (a) and (b) cannot occur at the same time

    $$\boxed{\neg a_i \vee \neg b_i}$$

  - There must be "at least one" action at each time step

    $$\boxed{a_i \vee b_i}$$

Note: assume that they are
the only two actions

Frame axiom

- Formulas describing "what doesn't change" between time steps (i) and (i+1)
  - There are several ways to write frame axioms

- One way: ***explanatory***
  - One axiom for each literal (l): which actions must be responsible for the change between time steps (i) and (i+1)

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

Example

- Robot planning
  - Robot (r1) and two locations (l1,l2)
  - One action at a time

move(r : robot, l : location, l' : location)
precond: at(r,l)
effects: at(r,l'), ¬at(r,l)

Example

- Robot planning
  - Robot (r1) and two locations (l1,l2)
  - One action at a time

move(r : robot, l : location, l' : location)
precond: at(r,l)
effects: at(r,l'), ¬at(r,l)

- Encoding (P,n) where n=1
  - Initial state:

    $$at(r1,l1,0) \wedge \neg at(r1,l2,0)$$

  - Goal state:

    $$at(r1,l2,1) \wedge \neg at(r1,l1,1)$$

  - Transition model:

    $$move(r1,l1,l2,0) \Rightarrow at(r1,l1,0) \wedge at(r1,l2,1) \wedge \neg at(r1,l1,1)$$
    $$move(r1,l2,l1,0) \Rightarrow at(r1,l2,0) \wedge at(r1,l1,1) \wedge \neg at(r1,l2,1)$$

Example (continued)

- Initial state:

  $at(r1,l1,0) \wedge \neg at(r1,l2,0)$

- Goal state:

  $at(r1,l2,1) \wedge \neg at(r1,l1,1)$

- Transition model:

  $move(r1,l1,l2,0) \Rightarrow at(r1,l1,0) \wedge at(r1,l2,1) \wedge \neg at(r1,l1,1)$

  $move(r1,l2,l1,0) \Rightarrow at(r1,l2,0) \wedge at(r1,l1,1) \wedge \neg at(r1,l2,1)$

- Complete exclusion axiom:

  $\neg move(r1,l1,l2,0) \vee \neg move(r1,l2,l1,0)$

  $move(r1,l1,l2,0) \vee move(r1,l2,l1,0)$

- Explanatory frame axiom:

  $\neg at(r1,l1,0) \wedge at(r1,l1,1) \Rightarrow move(r1,l2,l1,0)$

  $\neg at(r1,l2,0) \wedge at(r1,l2,1) \Rightarrow move(r1,l1,l2,0)$

  $at(r1,l1,0) \wedge \neg at(r1,l1,1) \Rightarrow move(r1,l1,l2,0)$

  $at(r1,l2,0) \wedge \neg at(r1,l2,1) \Rightarrow move(r1,l2,l1,0)$

High-level Procedure

function SATPLAN (*problem*, T_{max})
returns a solution, or failure
  inputs: *problem*, a planning problem
       T_{max}, an upper limit for plan length

  for $T = 0$ **to** T_{max} **do**
    cnf, *mapping* $\leftarrow$ TRANSLATE-TO-SAT(*problem*, T)
    assignment $\leftarrow$ SAT-SOLVER(*cnf*)
    if *assignment* is not null **then**
      return EXTRACT-SOLUTION(*assignment*,*mapping*)
  return *failure*

Outline for today

- **Solving AI Planning Problem using SAT**
  - **What is SAT?**
  - **How to implement a SAT solver?**
  - **How to reduce an AI planning problem to SAT?**