

Declarative problem solving

SAT, SMT & QBF and their applications in AI

Jussi Rintanen

NICTA and the Australian National University
Canberra, Australia

1. Formalization: synthesize a **formula** Λ that precisely describes how a **solution** looks like.
2. Find **one solution** or **all solutions**.
3. Different ways of reasoning about the solutions:
 - ▶ **Logical Consequence** $\Lambda \models \phi$ to test whether **all solutions** have property ϕ .
 - ▶ **Model-Finding** to find **one solution** of Λ
 - ▶ **Symbolic Methods** to represent and reason with **all solutions** of Λ .

Automated reasoning: application

Derivability / Logical consequence:

- ▶ Query answering $\Lambda \models \phi$.
- ▶ Properties ϕ shared by all solutions of Λ .
- ▶ If solutions differ from each other much, not many interesting ϕ .

Model-finding:

- ▶ Find **one model** or **all models**.
- ▶ Applications:
 - ▶ Find a **diagnosis** or **explanation**.
 - ▶ Find a **plan** [Kautz and Selman, 1992].
 - ▶ Find a **faulty behaviour** (CAV) [Biere et al., 1999]

Alternatives

Model-finding (as with SAT) is alternative to

1. **symbolic methods** which use **BDDs** or other normal forms,
2. **ad hoc** methods like **state-space search**.

Model-finding **advantages**:

- ▶ Finding one solution **more efficient** than with symbolic methods.
- ▶ **More flexible** than ad hoc methods.

Model-finding **disadvantages**:

- ▶ **All solutions** by enumeration only (vs. **compact representation** generated by general symbolic methods.)

Problem classes

- ▶ SAT: NP-complete problems (Boolean, discrete-valued)
- ▶ SMT: NP-complete problems, and harder ones (different arithmetic theories for example)
- ▶ QBF: PSPACE-complete problems, problems in the Polynomial Hierarchy
- ▶ Lots of (very difficult) problems outside these classes!

Propositional logic

Valuations and truth

Define truth with respect to a **valuation** $v : A \rightarrow \{0, 1\}$:

1. $v \models \top$
2. $v \not\models \perp$
3. $v \models a$ if and only if $v(a) = 1$, for all $a \in A$.
4. $v \models \neg\phi$ if and only if $v \not\models \phi$.
5. $v \models \phi \vee \phi'$ if and only if $v \models \phi$ or $v \models \phi'$.
6. $v \models \phi \wedge \phi'$ if and only if $v \models \phi$ and $v \models \phi'$.

Define for sets C of formulas, $v \models C$ iff $v \models \phi$ for all $\phi \in C$.

Propositional logic

Syntax

Let A be a set of atomic propositions (\sim state variables.)

1. \perp and \top are formulae.
2. a is a formula for all $a \in A$.
3. $\neg\phi$ is a formula if ϕ is.
4. $\phi \vee \phi'$ and $\phi \wedge \phi'$ are formulae if ϕ and ϕ' are.

$\phi \rightarrow \phi'$ is an abbreviation for $\neg\phi \vee \phi'$.

$\phi \leftrightarrow \phi'$ is an abbreviation for $(\phi \rightarrow \phi') \wedge (\phi' \rightarrow \phi)$.

For a literal $l \in A \cup \{\neg a \mid a \in A\}$ its **complement** \bar{l} is defined by $\bar{a} = \neg a$ and $\overline{\neg a} = a$.

A **clause** is a disjunction of literals $l_1 \vee \dots \vee l_n$.

The SAT decision problem

SAT

Let A be a set of propositional variables. Let \mathcal{F} be a set of clauses over A .

$\mathcal{F} \in \text{SAT}$ iff there is $v : A \rightarrow \{0, 1\}$ such that $v \models \mathcal{F}$.

UNSAT

Let A be a set of propositional variables. Let \mathcal{F} be a set of clauses over A .

$\mathcal{F} \in \text{UNSAT}$ iff $v \not\models \mathcal{F}$ for all $v : A \rightarrow \{0, 1\}$.

Complexity class NP

- ▶ **NP** = decision problems solvable by nondeterministic Turing machines with a polynomial bound on the number of computation steps
- ▶ This is roughly: search problems with a search tree (OR tree) of polynomial depth
- ▶ SAT is in NP because
 1. a valuation v of A can be guessed in $|A|$ steps, and
 2. testing $v \models \mathcal{F}$ is polynomial time in the size of \mathcal{F} .

Significance of NP-completeness

- ▶ No NP-complete problem is known to have a **polynomial time** algorithm.
- ▶ Best algorithms have a **worst-case exponential** runtime.
 - $2^{0.30897m}$, $2^{0.10299L}$ [Hirsch, 2000]
 - SAT: $(2 - \frac{2}{k+1})^n$ [Dantsin et al., 2002]
 - SAT: $2^{n(1 - \frac{1}{\ln \frac{m}{n} + O(\ln \ln m)})}$ [Dantsin et al., 2005]
 - (m clauses of length $\leq k$, n variables, size L).
- ▶ However, worst-case often doesn't show up!
- ▶ Current SAT algorithms solve problem instances with **millions of clauses** and **hundreds of thousands of propositional variables** in seconds.

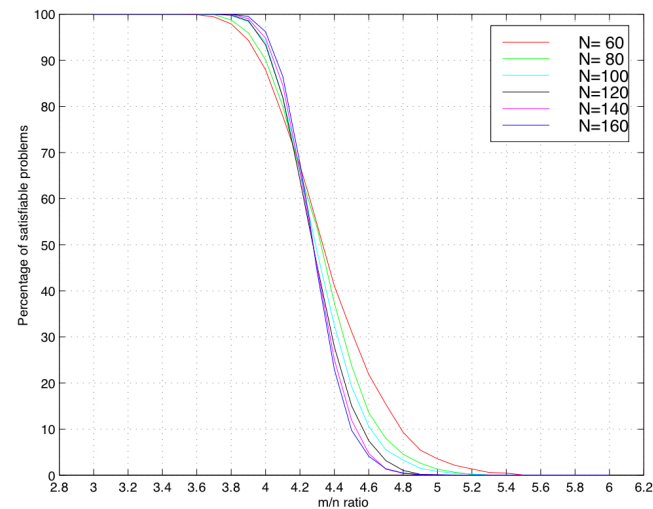
NP-hardness of SAT

[Cook, 1971]

- ▶ Cook showed that the halting problem of any **nondeterministic** Turing machine with a **polynomial time bound** can be reduced to SAT. Idea:
 - ▶ TM configuration \sim a valuation of propositional variables
 - ▶ sequence of configurations \sim sequence of valuations
 - ▶ relation between consecutive configurations \sim propositional formula
 - ▶ initial and accepting configurations \sim propositional formula
 - ▶ accepting computation \sim valuation that makes the formula true
- ▶ The proof is similar to the reduction from AI planning to SAT [Kautz and Selman, 1992]! We will discuss this in detail later.

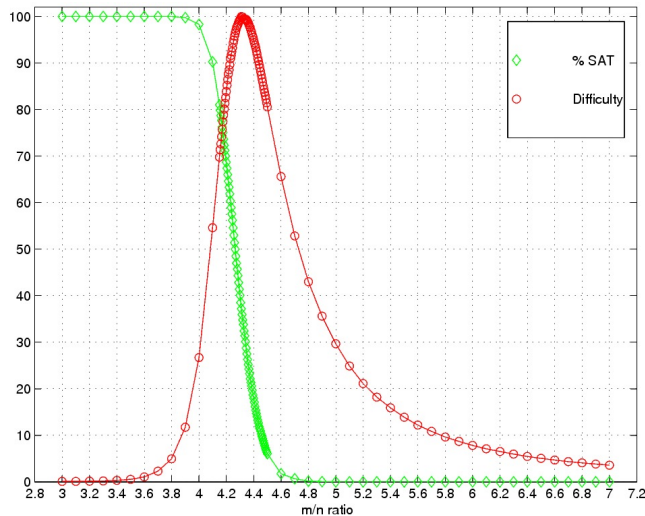
Phase transitions ([Mitchell et al., 1992])

Phase transition from SAT to UNSAT in 3-SAT



Phase transitions ([Mitchell et al., 1992])

Problem difficulty in the phase transition area



Meaning of phase transitions

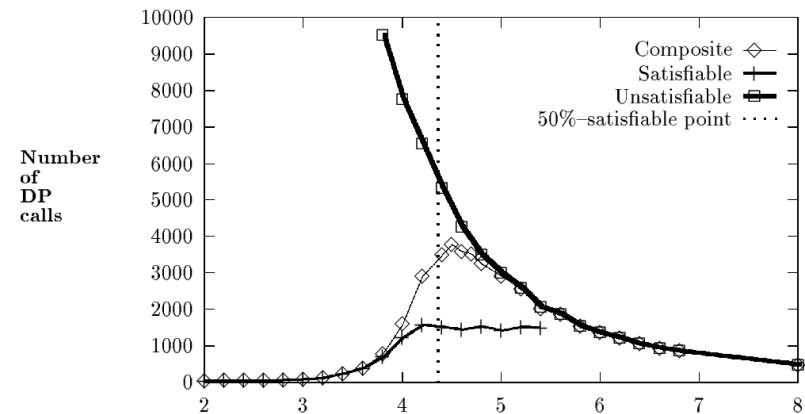
Even though all known complete algorithms have an exponential runtime in the **worst case**, their scalability on **under-constrained** and **over-constrained** problem instances is much better.

Other hard problems have similar phase transitions: keep problem size constant, and vary one of the parameters.

- ▶ **scheduling**: few..many tasks, a lot of..little time/resources
- ▶ **diagnosis**: few..many observations
- ▶ **planning**: many..few actions/events (cf. random graphs [Bollobás, 1985])

Phase transitions ([Mitchell et al., 1992])

Problem difficulty separately for SAT and UNSAT



Algorithms for SAT

- ▶ **Systematic algorithms**:
 - ▶ Keep track of the visited part of the search space.
 - ▶ Are in principle always able to answer *yes* or *no* to the $\phi \in \text{SAT}$ question.
 - ▶ Davis-Putnam-Logemann-Loveland procedure [Davis et al., 1962]
- ▶ **Local search algorithms**:
 - ▶ No memory of search history.
 - ▶ Won't terminate if no solution exists.
 - ▶ WalkSat [Selman et al., 1994], survey propagation [Mézard et al., 2002]

See <http://www.cril.univ-artois.fr/SAT09/> for the 2009 SAT competition.

CNF: Conjunctive Normal Form

A formula in **CNF** is a conjunction of clauses

$$(l_{1,1} \vee \dots \vee l_{1,n_1}) \wedge \dots \wedge (l_{m,1} \vee \dots \vee l_{m,n_m}).$$

1. Translation into **Negation Normal Form NNF** so that all negations are in front of atomic propositions:

$$\begin{aligned} \neg(\phi_1 \wedge \phi_2) &\equiv (\neg\phi_1 \vee \neg\phi_2) \\ \neg(\phi_1 \vee \phi_2) &\equiv (\neg\phi_1 \wedge \neg\phi_2) \\ \neg\neg\phi &\equiv \phi \end{aligned}$$

2. Moving conjunctions outside disjunctions (exponential!):

$$\begin{aligned} (\phi_1 \wedge \phi_2) \vee \phi_3 &\equiv (\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3) \\ \phi_3 \vee (\phi_1 \wedge \phi_2) &\equiv (\phi_1 \vee \phi_3) \wedge (\phi_2 \vee \phi_3) \end{aligned}$$

Truth-tables

Truth table for $\phi = (a \leftrightarrow b) \vee (c \rightarrow d)$:

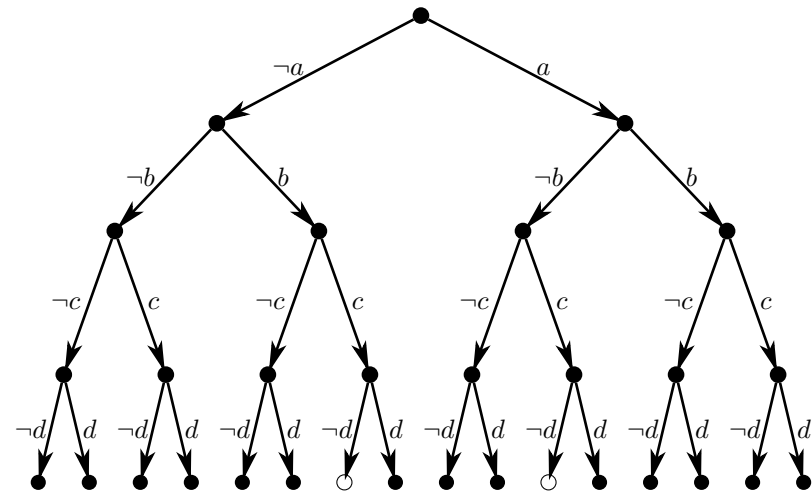
v				v(ϕ)
a	b	c	d	
0	0	0	0	1
0	0	0	1	1
0	0	1	0	1
0	0	1	1	1
0	1	0	0	1
0	1	0	1	1
0	1	1	0	0
0	1	1	1	1
1	0	0	0	1
1	0	0	1	1
1	0	1	0	0
1	0	1	1	1
1	1	0	0	1
1	1	0	1	1
1	1	1	0	1
1	1	1	1	1

Translation into CNF

- ▶ There are **satisfiability-preserving** translations into CNF with a **linear number of auxiliary variables** and only a **linear** (constant factor) size increase [Tseitin, 1962, Jackson and Sheridan, 2005, Manolios and Vroon, 2007].
- ▶ Some strong inference and simplification rules for circuits have no natural counterpart for CNF [Junttila and Niemelä, 2000].

Truth-tables vs binary search trees

Binary search tree for $\phi = (a \leftrightarrow b) \vee (c \rightarrow d)$:



The Resolution rule

Resolution

$$\frac{l \vee \phi \quad \bar{l} \vee \phi'}{\phi \vee \phi'}$$

One of l and \bar{l} is false.
Hence at least one of ϕ and ϕ' is true.

Unit propagation

Unit Resolution

$$\frac{l \quad \bar{l} \vee \phi}{\phi}$$

Unit Propagation algorithm $UP(\mathcal{F})$ for clause sets \mathcal{F}

1. If there is a **unit clause** $l \in \mathcal{F}$, then replace every $\bar{l} \vee \phi \in \mathcal{F}$ by ϕ and remove all non-unit clauses containing l from \mathcal{F} .
As a special case the **empty clause** \perp may be obtained.
2. Repeat step 1 until no more change takes place.
3. Return \mathcal{F} .

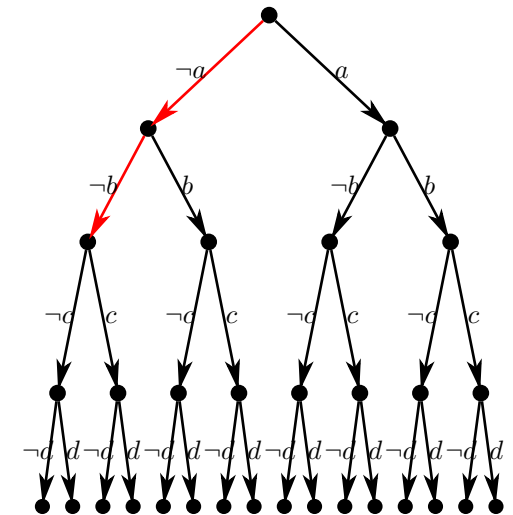
We sometimes write $\mathcal{F} \vdash_{UP} l$ if $l \in UP(\mathcal{F})$.

Unit Resolution

- ▶ Unrestricted application of the resolution rule is very expensive.
- ▶ **Unit resolution** restricts one of the clauses to be a **unit clause** consisting of only one literal.
- ▶ Performing all possible unit resolution steps on a clause set can be done in **linear time**.
- ▶ Unit Resolution is incomplete.

Binary search with unit resolution

$a \vee d$
 $b \vee c$



Davis-Putnam-Logemann-Loveland procedure

[Davis and Putnam, 1960, Davis et al., 1962]

- 1: *PROCEDURE* DPLL(C)
- 2: $C := UP(C)$;
- 3: *IF* $\{a, \neg a\} \subseteq C$ for some $a \in A$ *THEN RETURN* false;
- 4: $a :=$ any variable such that $\{a, \neg a\} \cap C = \emptyset$;
- 5: *IF* no such variable exists *THEN RETURN* true;
- 6: *IF* DPLL($C \cup \{a\}$) = true *THEN RETURN* true;
- 7: *RETURN* DPLL($C \cup \{\neg a\}$);

Clause Learning (CL)

- ▶ The Resolution rule is more powerful than DPLL: UNSAT proofs by DPLL may be **exponentially larger** than the smallest resolution proofs [Goerd, 1992].
- ▶ An extension to DPLL, based on **learning clauses**, is similarly exponentially more powerful than DPLL [Beame et al., 2004].
- ▶ For many applications SAT solvers with CL are the best.
- ▶ Also called conflict-driven clause learning (CDCL).

Additional rules for DPLL

Pure/monotonic literals

1. If $a \in A$ occurs only **positively**, set a true ($C := UP(C \cup \{a\})$.)
2. If $a \in A$ occurs only **negatively**, set a false.

Example

In $a \vee b, \neg b \vee \neg c, c$ the literal a is positive, and can be set true. After this, also $\neg b$ becomes a pure literal.

Failed literals

1. If $C \cup \{a\} \vdash_{UP} \perp$, set a false.
2. If $C \cup \{\neg a\} \vdash_{UP} \perp$ set a true.

Clause Learning (CL)

- ▶ Assume a partial assignment (a path in the DPLL search tree from the root to a leaf) corresponding to literals l_1, \dots, l_n leads to a contradiction (with unit resolution)

$$\mathcal{F} \cup \{l_1, \dots, l_n\} \vdash_{UP} \perp$$

From this follows

$$\mathcal{F} \models \bar{l}_1 \vee \dots \vee \bar{l}_n.$$

- ▶ Often not all of the literals l_1, \dots, l_n are needed for deriving the empty clause \perp , and a shorter clause is possible.

Top-Level CL Procedure

Example

level	literal	inferred literals	new branch level	literal	inferred liter
1	<i>a</i>	$x, \neg y$	1	<i>a</i>	$x, \neg y$
2	$\neg b$	z	2	$\neg b$	$z, \neg d$
3	<i>c</i>	$\neg v, w$			
4	<i>d</i>	$\neg p, \neg q, \perp$			

Conflict after setting *d*!

Learn clause $\neg x \vee \neg z \vee \neg d$.

Undo assignments *c* and *d*.

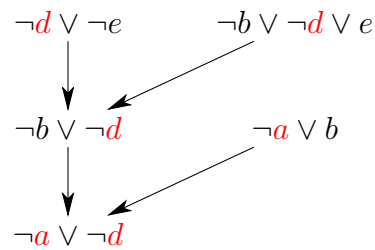
Unit resolution with x, z and the clause yields $\neg d$.

Clause Learning (CL)

Example

$\neg a \vee b$
 $\neg b \vee \neg d \vee e$
 $\neg d \vee \neg e$ falsified

a, b, c, d, e



Top-Level CL Procedure

- 1: *PROCEDURE* CL(\mathcal{F})
- 2: level := 0; X := \emptyset ;
- 3: *WHILE* $\mathcal{F} \cup X \not\vdash_{UP} \perp$ and there is an unassigned variable *DO*
- 4: level := level+1;
- 5: $l :=$ any unassigned literal;
- 6: $X := X \cup \{l\}$;
- 7: dl(l) := level;
- 8: *END*
- 9: *IF* $\mathcal{F} \cup X \vdash_{UP} \perp$ *THEN*
- 10: *IF* level = 0 *THEN RETURN* false;
- 11: $\phi :=$ learn(\mathcal{F}, X);
- 12: $\mathcal{F} := \mathcal{F} \cup \{\phi\}$;
- 13: level := max{dl(l) < level | $l \in \phi$ };
- 14: $X := \{l \in X \mid dl(l) < \text{level}\}$;
- 15: *GOTO* 3;
- 16: *RETURN* true;

Clause Learning (CL)

Procedure

The Reason of a Literal

For each non-decision literal l a **reason** is recorded: it is the clause $l \vee \phi$ from which l was derived with $\neg \phi$.

Learning One Clause (The Decision Scheme)

- ▶ Start with the clause $C = l_1 \vee \dots \vee l_n$ that was falsified.
- ▶ Resolve it with the reasons $\bar{l} \vee \phi$ of non-decision literals \bar{l} until only decision variables are left.

Learning One Clause

Different variants of the procedure

Decision Scheme Stop when only decision variables left.

First UIP (Unique Implication Point) Stop when only one literal of current decision level left.

Last UIP Stop when at the current decision level only the decision literal is left.

First UIP experimentally the best (if only one clause is learned.)
Some solvers learn more than one clause.

Heuristics for CL: VSIDS

Variable State Independent Decaying Sum [Moskewicz et al., 2001]

- ▶ Initially the score $s(l)$ of literal l is its number of occurrences in \mathcal{F} .
- ▶ When learned clause with l is added, increment $s(l)$.
- ▶ Periodically **decay** the scores by

$$s(l) := r(l) + 0.5s(l)$$

where $r(l)$ is the number of occurrences of l in learned clauses after the previous decay.

- ▶ Always choose unassigned literal l with maximum $s(l)$.

Newest version of zChaff and many other solvers use variants and extensions of VSIDS. The open-source **MiniSAT** solver decays 0.05 after **every** learned clause.

Research problem: Why do this kind of heuristics work so well?

Clause Learning (CL)

Forgetting clauses

- ▶ A problem with CL is **very high number** of learned clauses.
- ▶ Most SAT solvers **forget** clauses exceeding a length threshold in a regular interval to prevent the memory from filling up.

Heavy-tailed runtime distributions

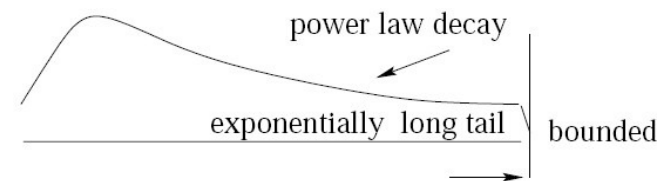


Diagram from [Chen et al., 2001]

On many types of problems the distributions characterize

- ▶ runtimes of a randomized algorithm on a single instance,
- ▶ runtimes of a deterministic algorithm on a class of instances.

Heavy-tailed runtime distributions

Estimating the mean is problematic

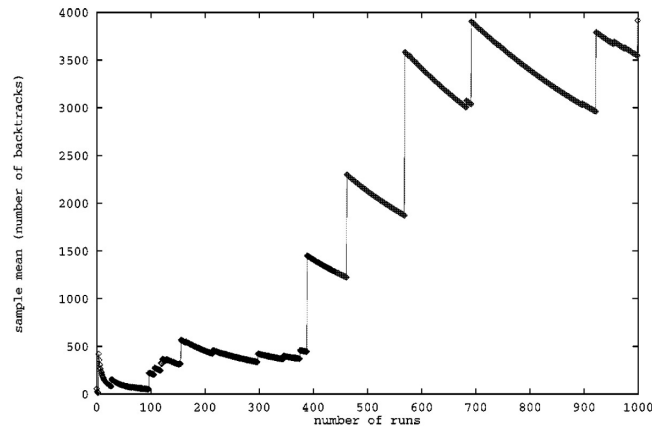


Diagram from [Gomes et al., 2000]

Restarts in SAT algorithms

Restarts were earlier used in stochastic local search algorithms:

- ▶ Necessary for **escaping local minima!**

[Gomes et al., 2000] demonstrated the utility of restarts for systematic SAT solvers:

- ▶ Small amount of **randomness** in branching variable selection.
- ▶ **Restart** the algorithm in regular intervals.

Heavy-tailed runtime distributions

Cause

- ▶ A small number of wrong decisions lead to a part of the search tree not containing any solutions.
- ▶ Backtrack-style search needs a long time to traverse the search tree.
 - ▶ Many short paths from the root node to a success leaf node.
 - ▶ High probability of reaching a huge subtree with no solutions.

These properties mean that

- ▶ average runtime is high,
- ▶ **restarting** the procedure after t seconds reduces the mean substantially, if t is close to the **mean of the original distribution**.

Restarts with Clause-Learning

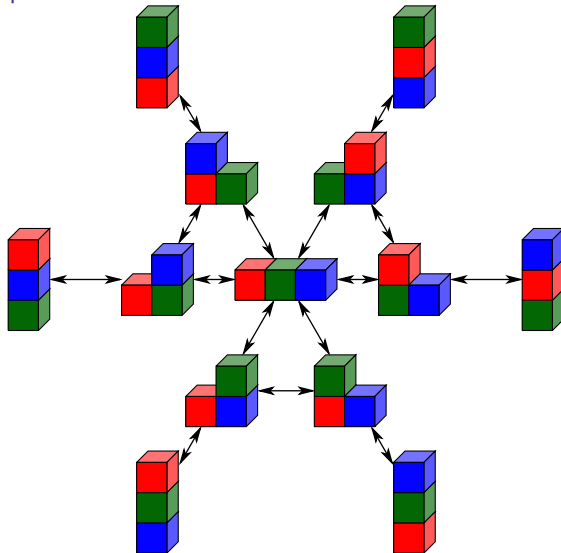
- ▶ Learned clauses are retained when doing the restart.
- ▶ **Problem:** Optimal restart policy depends on the runtime distribution, which is generally not known.
- ▶ **Problem:** Deletion of learned clauses and too early restarts may lead to **incompleteness**. Many implementations increase restart interval gradually.
- ▶ Paper: The effect of restarts on the efficiency of clause learning [Huang, 2007]

Applications of SAT

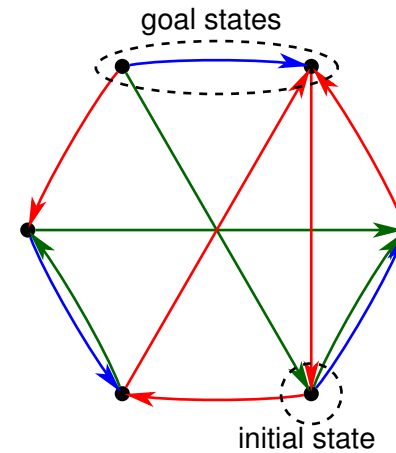
- ▶ Find a Path in a Transition Graph:
 - ▶ AI planning [Kautz and Selman, 1992]
 - ▶ computer-aided verification: model-checking [Biere et al., 1999]
 - ▶ diagnosis for discrete event systems [Grastien et al., 2007]
- ▶ many other verification problems
- ▶ haplotype inference in bioinformatics [Lynce and Marques-Silva, 2006]
- ▶ test-pattern generation [Larrabee, 1992]

Blocks world

The transition graph for three blocks



Transition systems



Paths with a given property

A central question about transition systems: is there a path satisfying a property P ?

application *different properties P*

planning (AI): last state satisfies goal G

model-checking (CAV): correctness property is violated

diagnosis: is compatible with observations; n faults

Succinct representation of transition systems

- ▶ state = valuation of a **finite set** of state variables

Example

HOUR : $\{0, \dots, 23\} = 13$

MINUTE : $\{0, \dots, 59\} = 55$

LOCATION : $\{51, 52, 82, 101, 102\} = 101$

WEATHER : $\{\text{sunny, cloudy, rainy}\} = \text{cloudy}$

HOLIDAY : $\{T, F\} = F$

- ▶ Any n -valued state variable can be represented by $\lceil \log_2 n \rceil$ Boolean (2-valued) state variables.
- ▶ Actions change the values of the state variables.

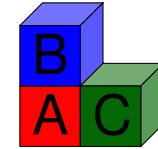
Formulae vs sets

sets	formulae
those $\frac{2^n}{2}$ states in which a is true	$a \in A \quad (n = A)$
$E \cup F$	$E \vee F$
$E \cap F$	$E \wedge F$
$E \setminus F$ (set difference)	$E \wedge \neg F$
\overline{E} (complement)	$\neg E$
the empty set \emptyset	\perp
the universal set	\top
question about sets	question about formulae
$E \subseteq F?$	$E \models F?$
$E \subset F?$	$E \models F$ and $F \not\models E?$
$E = F?$	$E \models F$ and $F \models E?$

Blocks world with Boolean state variables

Example

$s(\text{AonB})=0$ $s(\text{AonC})=0$ $s(\text{AonTABLE})=1$
 $s(\text{BonA})=1$ $s(\text{BonC})=0$ $s(\text{BonTABLE})=0$
 $s(\text{ConA})=0$ $s(\text{ConB})=0$ $s(\text{ConTABLE})=1$



Not all valuations correspond to an intended state: e.g. s' such that $s'(\text{AonB}) = 1$ and $s'(\text{BonA}) = 1$.

Sets (of states) as formulas

Formulae over A represent sets

$a \vee b$ over $A = \{a, b, c\}$

represents the **set** $\{010, 011, 100, 101, 110, 111\}$.

Formulae over $A \cup A'$ represent binary relations

$a \wedge a' \wedge (b \leftrightarrow b')$ over $A \cup A'$ where $A = \{a, b\}$, $A' = \{a', b'\}$

represents the **binary relation** $\{(10, 10), (11, 11)\}$.

Valuations 1010 and 1111 of $A \cup A'$ can be viewed respectively as **pairs of valuations** $(10, 10)$ and $(11, 11)$ of A .

Representation of one event/action

Change to one state variable

$e(a)$	$f_e(a)$
-	$a \leftrightarrow a'$
$a := 0$	$\neg a'$
$a := 1$	a'
$IF \phi THEN a := 1$	$(a \vee \phi) \leftrightarrow a'$
$IF \phi THEN a := 0$	$(a \wedge \neg \phi) \leftrightarrow a'$
$IF \phi_1 THEN a := 1;$ $IF \phi_0 THEN a := 0$	$(\phi_1 \vee (a \wedge \neg \phi_0)) \leftrightarrow a'$

A formula for one event/action e is now defined as

$$F(e) = \phi \wedge \bigwedge_{a \in A} f_e(a)$$

where $\phi = \text{prec}(e)$ is a **precondition** that has to be true for the event/action e to be possible.

Planning as a SAT problem

[Kautz and Selman, 1992]

- ▶ Problem is described by:
 - A a set of state variables
 - I, G formulas describing resp. the initial and goal states
 - O a set of actions
 - n plan length
- ▶ Generate a formula $\Phi_n^{I,O,G}$ such that
 - $\Phi_n^{I,O,G}$ is satisfiable
 - if and only if
 - there is a sequence of n actions in O from I to G .

Transition relations in the logic

A formula expressing the **choice** between events e_1, \dots, e_n is

$$\mathcal{T}(A, A') = \bigvee_{i=1}^n F(e_i).$$

We will later instantiate A and A' with different sets of propositional variables.

Planning with SAT

Components of an efficient planner:

1. Efficient **encoding** of planning in SAT
[Kautz and Selman, 1996, Rintanen et al., 2006].
2. Efficient **algorithm** for SAT.
3. Efficient **top-level algorithm** for finding plans
[Rintanen et al., 2006, Streeter and Smith, 2007].

Existence of plans of length n

Propositional variables

Define $A^i = \{a^i | a \in A\}$ for all $i \in \{0, \dots, n\}$.
 a^i expresses the value of $a \in A$ at time i .

Plans of length n in the propositional logic

Plans of length n correspond to satisfying valuations of

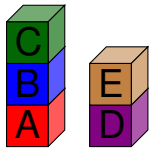
$$\Phi_n^{I,O,G} = I^0 \wedge \mathcal{T}(A^0, A^1) \wedge \mathcal{T}(A^1, A^2) \wedge \dots \wedge \mathcal{T}(A^{n-1}, A^n) \wedge G^n$$

where for any formula ϕ by ϕ^i we denote ϕ with propositional variables a replaced by a^i .

Planning as satisfiability

Example

initial state



goal state



Problem solved almost without search:

- ▶ Formulas for lengths 1 to 4 shown unsatisfiable without any search, only by unit resolution.
- ▶ Formula for plan length 5 is satisfiable: 3 nodes in the search tree.
- ▶ Plans have 5 to 7 actions, optimal plan has 5.

Planning as satisfiability

Example

Example

Consider

$$I = b \wedge c$$

$$G = (b \wedge \neg c) \vee (\neg b \wedge c)$$

$$o_1 = \text{IF } c \text{ THEN } \neg c; \text{ IF } \neg c \text{ THEN } c$$

$$o_2 = \text{IF } b \text{ THEN } \neg b; \text{ IF } \neg b \text{ THEN } b$$

Formula for plans of length 3 is

$$(b^0 \wedge c^0)$$

$$\wedge(((b^0 \leftrightarrow b^1) \wedge (c^0 \leftrightarrow \neg c^1)) \vee ((b^0 \leftrightarrow \neg b^1) \wedge (c^0 \leftrightarrow c^1)))$$

$$\wedge(((b^1 \leftrightarrow b^2) \wedge (c^1 \leftrightarrow \neg c^2)) \vee ((b^1 \leftrightarrow \neg b^2) \wedge (c^1 \leftrightarrow c^2)))$$

$$\wedge(((b^2 \leftrightarrow b^3) \wedge (c^2 \leftrightarrow \neg c^3)) \vee ((b^2 \leftrightarrow \neg b^3) \wedge (c^2 \leftrightarrow c^3)))$$

$$\wedge((b^3 \wedge \neg c^3) \vee (\neg b^3 \wedge c^3)).$$

Planning as satisfiability

Example

	0	1	2	3	4	5	0	1	2	3	4	5
clear(a)	FF	FF	FF	TT	FF	TTT	FFF	TT	FFF	TTT	TTT	TTT
clear(b)	F	F	FF	TT	FF	TTT	FF	TT	FFF	TTT	TTT	TTT
clear(c)	TT	FF	TTT	FF	TTT	FFF	TTT	FF	TTT	FFF	FFF	FFF
clear(d)	F	T	T	F	F	F	F	T	F	F	F	F
clear(e)	T	T	F	F	F	F	T	F	F	F	F	F
on(a,b)	FFF	T	FFF	FF	FFF	FFF	FFF	FFF	FFF	FFF	FFF	FFF
on(a,c)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(a,d)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(a,e)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(b,a)	TT	FF	TTT	FF	TTT	FFF	TTT	FF	TTT	FFF	FFF	FFF
on(b,c)	FF	TT	FFF	TT	FFF	FFF	FFF	TT	FFF	FFF	FFF	FFF
on(b,d)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(b,e)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(c,a)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(c,b)	T	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF
on(c,d)	FFF	TTT	FFF	TTT	FFF	TTT	FFF	TTT	FFF	TTT	FFF	TTT
on(c,e)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(d,a)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(d,b)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(d,c)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(d,e)	FF	TTT	FFF	TTT	FFF	TTT	FFF	TTT	FFF	TTT	FFF	TTT
on(e,a)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(e,b)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(e,c)	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF	FFFFF
on(e,d)	T	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF
on(e,e)	TTTT	F	TTTT	F	TTTT	F	TTTT	F	TTTT	F	TTTT	F
ontable(a)	TTT	F	FFF	FF	FFF	FF	FFF	FF	FFF	FF	FFF	FF
ontable(b)	FF	FF	FFF	FF	FFF	FF	FFF	FF	FFF	FF	FFF	FF
ontable(c)	F	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF	TT	FFF
ontable(d)	TTTT	FFF	TTTT	FFF	TTTT	FFF	TTTT	FFF	TTTT	FFF	TTTT	FFF
ontable(e)	F	TTTT	TTTT	FFF	TTTT	FFF	TTTT	FFF	TTTT	FFF	TTTT	FFF

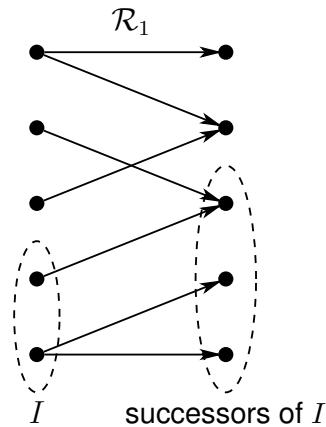
1. State variable values inferred from initial values and goals
2. Branch: $\neg \text{clear}(b)$ ¹
3. Branch: $\text{clear}(a)$ ³
4. Plan found:

	0	1	2	3	4
fromtable(a,b)	FFFF	T			
fromtable(b,c)	FFF	T	F		
fromtable(c,d)	FF	T	F	F	
fromtable(d,e)	F	T	F	F	F
totable(b,a)	F	T	F	F	F
totable(c,b)	F	T	F	F	F
totable(e,d)	T	F	F	F	F

Comparison: Planning with BDDs

$$\exists A^0.(I^0 \wedge \mathcal{T}(A^0, A^1))$$

projection of \mathcal{T} to time 1. This is the set of states that are reachable from I by taking one action.



DES Diagnosis

[Grastien et al., 2007]

- ▶ Action/event sequences of length n

$$\mathcal{T}(A^0, A^1) \wedge \mathcal{T}(A^1, A^2) \wedge \dots \wedge \mathcal{T}(A^{n-1}, A^n)$$

are directly applicable to **diagnosis** for **Discrete Event Systems DES**.

- ▶ Instead of I^0 and G^n , arbitrary **observations** (facts) about any time points can be expressed.
- ▶ Assumptions of 0, 1, 2 or more **faults** can be encoded by **cardinality constraints** [Bailleux and Boufkhad, 2003].
- ▶ Satisfying valuation \sim diagnosis.

Bounded LTL Model-checking [Biere et al., 1999]

= Planning with Temporally Extended Goals

- ▶ Testing **safety properties** is the same as AI planning: Goal $G \sim$ Safety property $\neg G$.
- ▶ Extension to Linear Temporal Logic **LTL** [Biere et al., 1999]: modular encoding of X, G, F, U over (infinite) state sequences.
- ▶ This is also **planning with temporally extended goals**.
- ▶ Efficient **linear size** encodings exist [Latvala et al., 2004].

SAT modulo Theories (SMT)

- ▶ Problem: SAT is insufficient for many applications
 - ▶ **continuous** resources (cost, distance, resources)
 - ▶ models with **real/rational time**
 - ▶ complex datatypes like trees, bit-vectors (anything with an **unbounded value domain**)
- ▶ Problem: first-order theorem proving is expensive.
- ▶ Idea: Have atomic propositions with **more structure** [Wolfman and Weld, 1999].

Examples:

- ▶ linear real/integer arithmetic
- ▶ real/integer difference logic
- ▶ bit vectors
- ▶ arrays

SAT modulo Theories (SMT)

Definition

SAT+ \mathcal{T} problem consists of a set of clauses $l_1 \vee \dots \vee l_n$ and $l'_1 \vee \dots \vee l'_m \vee \phi$ where l_i and l'_j are Boolean literals over some set A and ϕ is a proposition in the theory \mathcal{T} .

Example

Let $A = \{a, b, c\}$ and \mathcal{T} is Linear Real Arithmetic with variables $\{x, y, z\}$.

Then one SAT+ \mathcal{T} problem is:

$$\{a \vee b, (x + y = 2.5) \\ \neg b \vee (x - 2y \geq 0), \neg a \vee (x + 3z \geq 2)\}$$

One solution consists of assignments $v(a) = 1, v(b) = 0$ and $w(x) = 1.0, w(y) = 1.5, w(z) = 0.34$.

Algorithms for SMT

The “Lazy” Approach

1. Run DPLL assigning values to Boolean variables only.
2. When no Boolean variables left, solve the remaining theory with a solver for the theory.
3. If no solution, backtrack.

Enhancements:

- ▶ Early pruning: Run the theory solver as soon as some non-Boolean propositions are unit, and backtrack if not satisfiable.

Arithmetics

theory	complexity	algorithms
Linear Real Arithmetics LRA	poly-time	
LRA + max/minimization (LP)	poly-time	Simplex
Linear Integer Arithmetics LIA	NP-hard	

DPLL+ \mathcal{T}

Let $\text{SOLVE}_{\mathcal{T}}(T)$ be a solver for the theory \mathcal{T} , which returns *true* or *false* depending on the satisfiability of T .

- 1: *PROCEDURE* DPLL+ $\mathcal{T}(C, \mathcal{T})$
- 2: $C := \text{UP}(C)$;
- 3: *IF* $\{a, \neg a\} \subseteq C$ for some $a \in A$ *THEN RETURN* false;
- 4: *IF* there is a variable such that $\{a, \neg a\} \cap C = \emptyset$ *THEN*
- 5: *IF* DPLL($C \cup \{a\}$) = true *THEN RETURN* true;
- 6: *RETURN* DPLL($C \cup \{\neg a\}$);
- 7: *ELSE*
- 8: *RETURN* $\text{SOLVE}_{\mathcal{T}}(T)$;

Practical improvements to the \mathcal{T} solver

Monotonicity of the theory on a given branch

- ▶ When proceeding a search tree branch, the theory T **monotonically grows**.
- ▶ Therefore, information about T can be **reused** when solving $T \cup T'$.

Efficient backtracking

- ▶ Upon backtracking, the current theory $T \cup T'$ becomes the smaller T .
- ▶ **Restore** the state T quickly.

Application: Timed Systems

- ▶ events or actions with a **duration**: change is not caused immediately, but after a known time interval.
- ▶ Timed models common in
 - ▶ computer aided verification and validation [Audemard et al., 2002],
 - ▶ diagnosis (abduction, explanation),
 - ▶ planning and control.

SMT implementations

Early implementation: LPSAT [Wolfman and Weld, 1999]

solver	theory	reference
MathSAT	LRA, LIA, BV	[Bozzano et al., 2005]
CVC	LRA, LIA, BV	[Barrett et al., 2002]
ICS		[de Moura et al., 2002]
Yices	LRA, LIA	(Dutertre and de Moura, 2002-)
Z3.2	LRA, LIA, BV	[de Moura and Bjørner, 2008]
Barcelogic	LRA, LIA	[Bofill et al., 2008]

See <http://www.smtcomp.org/> for SMT Competitions (2008, 2009).

Representation of timed systems

- ▶ Use SMT instead of SAT.
- ▶ represented time points: anywhere where something happens
- ▶ continuous variables are used for counting time (compare **timed automata** [Alur and Dill, 1994])

Temporal Planning

Events/actions e :

Precondition (a non-temporal formula)

Effects $L@t$ where L is a set of literals and $t > 0$ is a time.

If action takes place at t' , then effects L take place at $t' + t$.

Temporal Planning

SAT+LRA encoding

problem instance: $A = \{a^1, \dots, a^m\}$ $E = \{e^1, \dots, e^k\}$

Time horizon is $n + 1$ points with change.

Variables in the encoding (for all $j \in \{0, \dots, n\}$) are:

T_0, \dots, T_n absolute time at time points $0, \dots, n$.

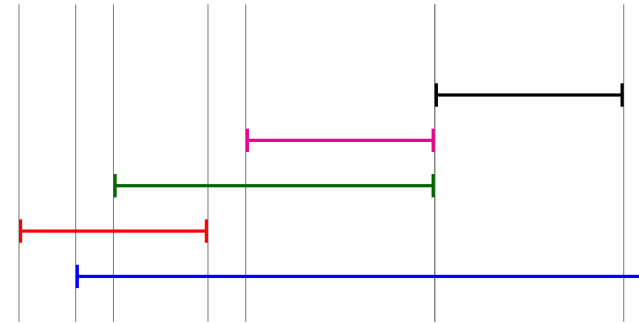
e_j event $e \in E$ takes place at $j \in \{0, \dots, n - 1\}$.

$c_j^{e,L@t}$ counter for each event, effect pair $(e, L@t)$.

Temporal Planning

SAT+LRA encoding

Idea: represent only time points with change or event/action.



Temporal Planning

SAT+LRA encoding

Precondition axioms: $e_i \rightarrow \phi_i$.

Effect axioms: updates to the counter for $L@t$ of e :

$$e_j \rightarrow (c_j^{e,L@t} = t)$$

$$\neg e_j \rightarrow (c_j^{e,L@t} = c_{j-1}^{e,L@t} - (T_j - T_{j-1}))$$

Effects of an event/action e with $L@t$:

$$(c_j^{e,L@t} = 0) \rightarrow \bigwedge L_j$$

Frame axioms for every state variable a :

$$(a_j \wedge \neg a_{j+1}) \rightarrow ((c_j^{e_1, L_1 @ t_1} = 0) \vee \dots \vee (c_j^{e_m, L_m @ t_n} = 0))$$

where $e_i/L_i@t_i, i \in \{1, \dots, n\}$ are all the events/effects with $a \in L_i$.

Relation between T_j and T_{j+1} : time may not pass a point where something happens.

$$(c_j^{e,L@t} > 0) \rightarrow (T_{j+1} \leq T_j + c_j^{e,L@t})$$

$$(T_j < T_{j+1})$$

What are hybrid systems?

- ▶ **continuous state variables** with complex **time-dependent** change (physical phenomena)
- ▶ The value of a variable changes continuously over time.
- ▶ Discrete events separate continuous value intervals.
- ▶ All main reasoning problems arise for hybrid systems:
 - ▶ computer aided verification and validation
 - ▶ diagnosis (abduction, explanation)
 - ▶ planning and control
- ▶ formal models: **hybrid automata** [Alur et al., 1993]

Hybrid systems in SMT

- ▶ State described by **discrete** (Boolean) and **continuous** variables.
- ▶ The discrete state determines the **evolution** of the continuous variables.
- ▶ Evolution of a continuous variable described by an **arithmetic equation**.

Reasoning about hybrid system

- ▶ Problems about **hybrid systems**, an extension of timed systems with **continuous change**, can be expressed in SMT.
- ▶ Earlier works include **model-checking** of hybrid systems [Giorgetti et al., 2005, Audemard et al., 2005].
- ▶ Obvious extensions to **planning** and **diagnosis** for **hybrid systems** possible.

Our Formalization

- ▶ Actions don't directly change the values of the continuous variables.
- ▶ Actions change the values of Boolean/discrete variables, which determine the current **law of change** for each continuous variable.
- ▶ Each continuous variable changes according to its current law until the condition for the law does not hold any more, and some other law is used instead.

Our Formalization

Laws of change are expressed as follows.

condition	change
$\phi^{x,1}$	$x(t + \Delta t) = f^{x,1}(x(0), \Delta t)$
$\phi^{x,2}$	$x(t + \Delta t) = f^{x,2}(x(0), \Delta t)$
\vdots	\vdots
ϕ^{x,n_x}	$x(t + \Delta t) = f^{x,n_x}(x(0), \Delta t)$

A requirement is that the formulae $\phi^{x,i}$ are mutually exclusive and logically exhaustive: $\phi^i \wedge \phi^j$ is inconsistent for any i and j such that $i \neq j$, and $\bigwedge_{i=1}^{n_x} \phi^{x,i} \equiv \top$.

Encoding of Change for Hybrid Systems

The change in the continuous variable x between steps j and $j + 1$ is encoded by the following formulae for every law $\phi^{x,i}, f^{x,i}$.

$$\phi^{x,i} \rightarrow x_{j+1} = f^{x,i}(x_j, T_{j+1} - T_j)$$

Our Formalization

To obtain compact encodings, change must be splittable.

$$f(x, \Delta t_1 + \Delta t_2) = f(f(x, \Delta t_1), \Delta t_2).$$

This property is satisfied by

- ▶ $f(x, \Delta t) = x + c\Delta t$: linear change proportional to t
- ▶ $f(x, \Delta t) = x \cdot r^{c\Delta t}$: exponential change
- ▶ $f(x, \Delta t) = c$: new constant value
- ▶ $f(x, \Delta t) = x$: no change, previous value

Defined (dependent) continuous variables can be obtained by arbitrary arithmetic combinations of the primitive variables.

Quantified Boolean formulae: motivation

- ▶ Where SAT is the logic of OR-trees (find one satisfying valuation), QBF is the logic of AND-OR trees.
- ▶ Some applications can be seen as 2-player games:
 - ▶ planning under uncertainty: agent vs. environment
 - ▶ model-checking CTL: path-quantifiers \forall and \exists .

The QBF decision problem

QBF

Let $A = \{x_1, \dots, x_n\}$ be a set of propositional variables. Let ϕ be a formula over A .

Let $\pi = Q_1x_1Q_2x_2 \cdots Q_nx_n$ be a **prefix** with each Q_i either the **existential** quantifier \exists or the **universal** quantifier \forall .

Then $\pi\phi$ is a **quantified Boolean formula**.

The QBF decision problem

$\mathcal{F} \in \text{TRUE-QBF}$ iff the formula $QE(\pi\phi)$ is a valid formula.

$$QE(\exists x\phi) = \phi[\top/x] \vee \phi[\perp/x]$$

$$QE(\forall x\phi) = \phi[\top/x] \wedge \phi[\perp/x]$$

$$QE(\phi) = \phi \text{ for any } \phi \text{ which does not start with } \exists \text{ or } \forall$$

More permissive definitions of QBF

- ▶ Some solution methods depend on a **non-prenex** representation.
- ▶ Instead of $\forall a \exists x \exists y. (a \vee x) \wedge (a \vee y)$, one could write $\forall a. ((\exists x. a \vee x) \wedge (\exists y. a \vee y))$.
- ▶ Making the scopes as small as possible makes dependencies between variables more explicit.

Quantified Boolean formulae

Definition

- ▶ If ϕ is a propositional formula and σ is a sequence of $\exists p$ and $\forall p$, one for every $p \in A$, then $\sigma\phi$ is a QBF.
- ▶ $\exists x\phi$ is true if and only if $\phi[\top/x] \vee \phi[\perp/x]$ is true.
- ▶ $\forall x\phi$ is true if and only if $\phi[\top/x] \wedge \phi[\perp/x]$ is true.

Example

The formulae $\forall x \exists y (x \leftrightarrow y)$ and $\exists x \exists y (x \wedge y)$ are true.

The formulae $\exists x \forall y (x \leftrightarrow y)$ and $\forall x \forall y (x \vee y)$ are false.

QBF roughly corresponds to search problems with AND-OR search trees of polynomial depth: $\text{APTIME} = \text{PSPACE}$.

Quantified Boolean formulae

QBF vs SAT

The **evaluation problem of QBF** generalizes both the **satisfiability** and **validity/tautology problems** of the propositional logic.

$\exists x_1 \cdots \exists x_n \Phi(x_1, \dots, x_n)$ is **true** iff $\Phi(x_1, \dots, x_n)$ is satisfiable.

$\forall x_1 \cdots \forall x_n \Phi(x_1, \dots, x_n)$ is **true** iff $\Phi(x_1, \dots, x_n)$ is valid.

Complexity

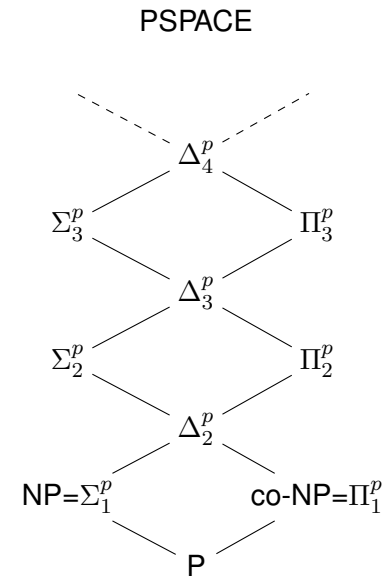
prefix	complexity
\exists	NP-complete
\forall	co-NP-complete
$\exists\forall$	Σ_2^p -complete
$\forall\exists$	Π_2^p -complete

Prefix with n alternations starting with \exists is Σ_n^p -complete.
 Prefix with n alternations starting with \forall is Π_n^p -complete.

Algorithms for QBF

- ▶ **Q-resolution** (generalization of the resolution rule) [Kleine Büning et al., 1995]
- ▶ **Q-DPLL** [Cadoli et al., 2002, Rintanen, 1999b]
- ▶ **clause learning** (with DPLL) [Zhang and Malik, 2002]
- ▶ **quantifier elimination** [Biere, 2004]

The Polynomial Hierarchy



Q-resolution for QBF-CNF

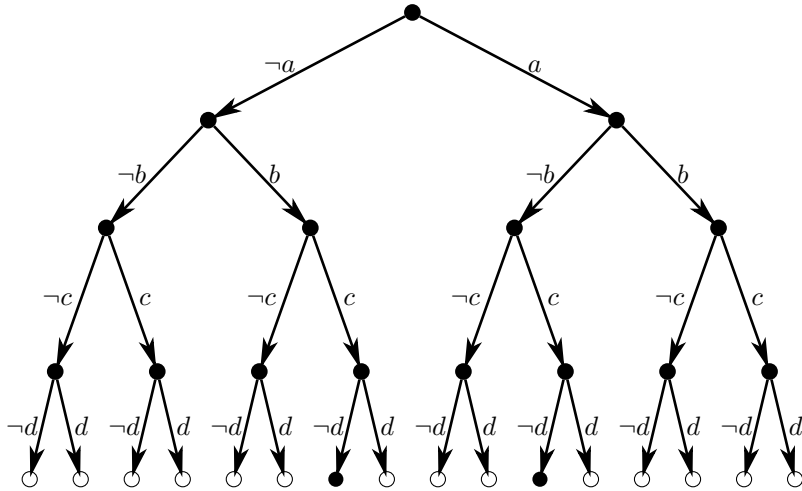
[Kleine Büning et al., 1995]

Inference rules

- ▶ Pure \forall -clauses can be replaced by the empty clause.
- ▶ If a clause contains a \forall -literal l that does not precede an \exists -literal in the clause, then remove l .
- ▶ Let x be an \exists -variable.
 Let $x \vee \phi$ and $\neg x \vee \phi'$ be two clauses so that ϕ and ϕ' don't contain complementary literals.
 Q-resolution lets us derive $\phi \vee \phi'$.

Binary search trees for QBF

Binary search tree for $\exists a \forall b \exists c \exists d ((a \leftrightarrow b) \vee (c \rightarrow d))$:



Clause Learning for Q-DPLL

[Letz, 2002, Zhang and Malik, 2002, Giunchiglia et al., 2006]

- ▶ Learning for both **falsifying** and **satisfying** valuations.
- ▶ Falsifying valuations: learn a clause as in SAT.
- ▶ Satisfying valuations: learn a **cube** = a **conjunction of literals**.

The body of the QBF during search has the form $\sigma \vee \Phi$, where

- ▶ σ is a **disjunction of cubes** and
- ▶ Φ is a conjunction of clauses (including the input CNF.)

Learned clauses and cubes work in a dual way: a cube justifies skipping satisfying parts of the search tree exactly like a clause does for unsatisfying parts.

Q-DPLL for QBF-CNF

```

1: PROCEDURE Q-DPLL( $e, \langle M_1, M_2, \dots, M_n \rangle, C$ )
2: BEGIN
3:    $C := UP(C)$ ;
4:   IF  $\{a, \neg a\} \subseteq C$  for some  $a \in A$  THEN RETURN false;
5:   IF  $n = 0$  THEN RETURN true;
6:   IF  $M_1 = \emptyset$  THEN RETURN Q-DPLL(not  $e, \langle M_2, \dots, M_n \rangle, C$ );
7:    $x :=$  any member of  $M_1$ ;
8:   IF  $e$  THEN                                     (*  $\exists$  variable, OR-node *)
9:     IF Q-DPLL( $e, \langle M_1 \setminus \{x\}, \dots, M_n \rangle, C \cup \{x\}$ )
10:    THEN RETURN true;
11:  ELSE                                           (*  $\forall$  variable, AND-node *)
12:    IF not Q-DPLL( $e, \langle M_1 \setminus \{x\}, \dots, M_n \rangle, C \cup \{x\}$ )
13:    THEN RETURN false;
14:  RETURN Q-DPLL( $e, \langle M_1 \setminus \{x\}, \dots, M_n \rangle, C \cup \{\neg x\}$ )
15: END

```

Quantifier/Variable Elimination

Algorithm by [Biere, 2004] eliminates variables.

- ▶ **Eliminate** (innermost) variables one by one.
- ▶ When only \exists -variables left, **call a SAT solver**.

Variables eliminated by

- ▶ \forall : equivalence $\forall x.\phi \equiv \phi[\top/x] \wedge \phi[\perp/x]$
- ▶ \exists : Q-resolution or equivalently $\exists x.\phi \equiv \phi[\top/x] \vee \phi[\perp/x]$
- ▶ simplifications that eliminate occurrences of \top, \perp

Solvers

- ▶ **QuBE** based on Q-DPLL and clause-learning [Giunchiglia et al., 2006].
- ▶ **Quantor** [Biere, 2004] eliminates quantifiers and uses
 - ▶ Equivalence reasoning: clauses $l \vee l'$ and $\bar{l} \vee \bar{l}'$ imply $l \leftrightarrow \bar{l}'$. Replace all occurrences of l by \bar{l}' .
 - ▶ Subsumption: if $\phi \in C$, then remove any clause $\phi \vee \psi \in C$.
- ▶ **Skizzo** [Benedetti, 2005] eliminates \exists -variables and represents the resulting clause set compactly, which is searched by a DPLL type algorithm. (Benedetti calls his \exists -elimination “symbolic Skolemization”, but it is equivalent to $\exists x.\phi \equiv \phi[\top/x] \vee \phi[\perp/x]$.)

Conformant planning ~ no observability

- ▶ Find a **sequence of actions** that reach the goals **under all contingencies**.
- ▶ Special case of **partial observability**.
- ▶ Main algorithms generalize to partial observability.

Applications of QBF

- ▶ AI planning generalized to **nondeterministic** domains, with a small (polynomial-size) plan (Σ_2^P -complete)
- ▶ very **compact representations** of some classes of SAT problems, for example, planning with **exponentially long plans** (PSPACE-complete)
- ▶ **LTL model-checking** generalized to **CTL*** (PSPACE-complete)

QBF for conditional planning

- ▶ [Rintanen, 1999a] introduced QBF to nondeterministic planning: problem not in NP but Σ_2^P -complete, must use QBF.
- ▶ Prefix $\exists\forall\exists$ says **there is** a plan such that **for all** contingencies **there is** an execution achieving the goals. (Ferraris & Giunchiglia 2000) split this to **search** over candidate plans + **validity test** (with SAT) for candidate plans. Conformant-FF (Brafman-Hoffmann 2003) does the same.

Encoding of nondeterminism with \forall -variables

Example

$e_1 = \langle \neg a, \overbrace{(b|d)}^{x_{11}} \wedge \overbrace{(a|c)}^{x_{12}} \rangle$ and $e_2 = \langle \neg b, \overbrace{((b|c)|a)}^{x_2} \rangle$ translate to

$$\begin{aligned} \neg(a \wedge \neg a') & \quad (\neg a \wedge a') \rightarrow ((e_1 \wedge x_{12}) \vee (e_2 \wedge \neg x_2)) \\ \neg(b \wedge \neg b') & \quad (\neg b \wedge b') \rightarrow ((e_1 \wedge x_{11}) \vee (e_2 \wedge x_2 \wedge x_{21})) \\ \neg(c \wedge \neg c') & \quad (\neg c \wedge c') \rightarrow ((e_1 \wedge \neg x_{12}) \vee (e_2 \wedge x_2 \wedge \neg x_{21})) \\ \neg(d \wedge \neg d') & \quad (\neg d \wedge d') \rightarrow (e_1 \wedge \neg x_{11}) \end{aligned}$$

$$\begin{aligned} e_1 & \rightarrow \neg a \\ (e_1 \wedge x_{11}) & \rightarrow b' & (e_1 \wedge x_{12}) & \rightarrow a' \\ (e_1 \wedge \neg x_{11}) & \rightarrow d' & (e_1 \wedge \neg x_{12}) & \rightarrow c' \\ e_2 & \rightarrow \neg b \\ (e_2 \wedge x_2 \wedge x_{21}) & \rightarrow b' & (e_2 \wedge \neg x_2) & \rightarrow a' \\ (e_2 \wedge x_2 \wedge \neg x_{21}) & \rightarrow c' \end{aligned}$$

Unbounded planning

Is there a plan of length 2^n ?

$$\exists S_0 \exists S_1 (\text{reach}_{2^n}(S_0, S_1) \wedge I^0 \wedge G^1)$$

where

$$\begin{aligned} \text{reach}_1(S, S') & \equiv \mathcal{R}(S, S') \\ \text{reach}_{2^i}(S, S') & \equiv \exists T \forall b \exists T_1 \exists T_2 (\text{reach}_i(T_1, T_2) \\ & \quad \wedge (b \rightarrow (T_1 = S \wedge T_2 = T)) \\ & \quad \wedge (\neg b \rightarrow (T_1 = T \wedge T_2 = S'))) \end{aligned}$$

Conformant planning in QBF

This is the standard encoding [Rintanen, 2007].

$$\exists P \forall C \exists D \left(I^0 \rightarrow \left(\bigwedge_{i=0}^{t-1} T(i, i+1) \wedge G^t \right) \right) \quad (1)$$

where

$$P = \bigcup_{i=0}^{t-1} O^i \quad C = A^0 \cup \bigcup_{i=0}^{t-1} X_i \quad D = \bigcup_{i=1}^t A^i$$

and

- ▶ Variables in A^i represent the state variables at time i .
- ▶ Variables in O^i represent the events/actions at i .
- ▶ Variables in X^i represent the contingencies at i .

Stochastic satisfiability SSAT

[Littman et al., 2001]

- ▶ SSAT extends QBF with a third quantifier \mathcal{R} for **random choice**. It takes the (weighted) **average** (in the same sense \exists takes the **maximum** and \forall the **minimum**), and the decision problem is to test whether the value of the formula exceeds a given threshold $\theta \in [0, 1]$.
- ▶ Special case: a propositional formula ϕ quantified with only $\mathcal{R}^{0.5}$ as $\mathcal{R}^{0.5} x_1 \cdots \mathcal{R}^{0.5} x_n (E[\phi] \geq \theta)$, is true if the proportion of valuations that satisfy ϕ is at least θ .
- ▶ Problem with SSAT: random variables are assumed to be independent.

Stochastic satisfiability SSAT



The value of a formula is defined as follows.

1. If the prefix is empty, $\text{val}(\phi) = \begin{cases} 1 & \text{if } \phi \equiv \top \\ 0 & \text{otherwise.} \end{cases}$
2. $\text{val}(\forall x \phi) = \min(\text{val}(\phi[\top/x]), \text{val}(\phi[\perp/x]))$
3. $\text{val}(\exists x \phi) = \max(\text{val}(\phi[\top/x]), \text{val}(\phi[\perp/x]))$
4. $\text{val}(\mathcal{R}^p x \phi) = p \cdot \text{val}(\phi[\top/x]) + (1 - p) \cdot \text{val}(\phi[\perp/x])$

A formula $Q_1 x_1 \cdots Q_n x_n (E[\phi] \geq \theta)$ is *true* if and only if

$$\text{val}(Q_1 x_1 \cdots Q_n x_n \phi) \geq \theta.$$

Bibliography I

-  Alur, R., Courcoubetis, C., Henzinger, T. A., and Ho, P.-H. (1993). Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*, pages 209–229. Springer-Verlag.
-  Alur, R. and Dill, D. L. (1994). A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235.
-  Audemard, G., Bozzano, M., Cimatti, A., and Sebastiani, R. (2005). Verifying industrial hybrid systems with MathSAT. *Electronic Notes in Theoretical Computer Science*, 119(2):17–32.
-  Audemard, G., Cimatti, A., Kornilowicz, A., and Sebastiani, R. (2002). Bounded model checking for timed systems. In *Formal Techniques for Networked and Distributed Systems - FORTE 2002*, number 2529 in *Lecture Notes in Computer Science*. Springer-Verlag.
-  Bailleux, O. and Boufkhad, Y. (2003). Efficient CNF encoding of Boolean cardinality constraints. In Rossi, F., editor, *Principles and Practice of Constraint Programming – CP 2003: 9th International Conference, Kinsale, Ireland, September 29 – October 3, 2003, Proceedings*, volume 2833 of *Lecture Notes in Computer Science*, pages 108–122. Springer-Verlag.

Probabilistic planning in SSAT

[Majercik and Littman, 2003]

- ▶ The quantifier \mathcal{R}^p can be used for expressing probability distributions over
 1. the initial states,
 2. the successor states of a nondeterministic action, or
 3. noisy observations.





The threshold θ can express a required **success probability** of a plan.

- ▶ Majercik & Littmann show how **conditional planning** can be encoded with SSAT with prefix






$$\exists \mathcal{R} \exists \mathcal{R} \cdots \exists$$

Each block of **existential** variables encodes a **choice** of actions conditional on previous observations, and each block of **random** variables encodes a **random outcome** of the previous action.

Bibliography II






-  Barrett, C. W., Dill, D. L., and Stump, A. (2002). Checking satisfiability of first-order formulas by incremental translation to SAT. In *Computer Aided Verification*, volume 2404 of *Lecture Notes in Computer Science*, pages 681–710. Springer-Verlag.
-  Beame, P., Kautz, H., and Sabharwal, A. (2004). Towards understanding and harnessing the potential of clause learning. *Journal of Artificial Intelligence Research*, 22:319–351.
-  Benedetti, M. (2005). Evaluating QBFs via symbolic Skolemization. In *Logic for Programming, Artificial Intelligence, and Reasoning*, number 3452 in *Lecture Notes in Computer Science*, pages 285–300. Springer-Verlag.
-  Biere, A. (2004). Resolve and expand. In *SAT 2004 - The Seventh International Conference on Theory and Applications of Satisfiability Testing, 10-13 May 2004, Vancouver, BC, Canada, Online Proceedings*, pages 59–70.

Bibliography III

-  **Biere, A., Cimatti, A., Clarke, E. M., and Zhu, Y. (1999).**
Symbolic model checking without BDDs.
In Cleaveland, W. R., editor, *Tools and Algorithms for the Construction and Analysis of Systems, Proceedings of 5th International Conference, TACAS'99*, volume 1579 of *Lecture Notes in Computer Science*, pages 193–207. Springer-Verlag.
-  **Bofill, M., Nieuwenhuis, R., Oliveras, A., Rodríguez-Carbonell, E., and Rubio, A. (2008).**
The Barcelogic SMT solver (tool paper).
In *Computer Aided Verification*, volume 5123 of *Lecture Notes in Computer Science*, pages 294–298. Springer-Verlag.
-  **Bollobás, B. (1985).**
Random graphs.
Academic Press.
-  **Bozzano, M., Bruttomesso, R., Cimatti, A., Junttila, T., van Rossum, P., Schulz, S., and Sebastiani, R. (2005).**
The MathSAT 3 system.
In *Automated Deduction - CADE-20*, volume 3632 of *Lecture Notes in Computer Science*, pages 315–321. Springer-Verlag.
-  **Cadoli, M., Schaerf, M., and Giovanardi, A. (2002).**
An algorithm to evaluate quantified Boolean formulae and its experimental evaluation.
Journal of Automated Reasoning, pages 101–142.





105 / 114

Bibliography V

-  **Davis, M. and Putnam, H. (1960).**
A computing procedure for quantification theory.
Journal of the ACM, 7(3):201–215.
-  **de Moura, L. and Bjørner, N. (2008).**
Z3: An efficient SMT solver.
In *Tools and Algorithms for the Construction and Analysis of Systems*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer-Verlag.
-  **de Moura, L., Rueß, H., and Sorea, M. (2002).**
Lazy theorem proving for bounded model checking over infinite domains.
In *Automated Deduction CADE-18*, volume 2392 of *Lecture Notes in Computer Science*, pages 1–4. Springer-Verlag.
-  **Giorgetti, N., Pappas, George, J., and Bemporad, A. (2005).**
Bounded model checking of hybrid dynamical systems.
In *Proceedings of the 44th IEEE Conference on Decision and Control, and the European Control Conference 2005*, pages 672–677. IEEE.
-  **Giunchiglia, E., Narizzano, M., and Tacchella, A. (2006).**
Clause/term resolution and learning in the evaluation of quantified Boolean formulas.
Journal of Artificial Intelligence Research, 26:371–416.






107 / 114

Bibliography IV

-  **Chen, H., Gomes, C., and Selman, B. (2001).**
Formal models of heavy-tailed behavior in combinatorial search.
In Walsh, T., editor, *Proceedings of the 7th International Conference on Principles and Practice of Constraint Programming*, number 2239 in *Lecture Notes in Computer Science*, pages 408–421. Springer-Verlag.
-  **Cook, S. A. (1971).**
The complexity of theorem proving procedures.
In *Proceedings of the Third Annual ACM Symposium on Theory of Computing*, pages 151–158.
-  **Dantsin, E., Goerd, A., Hirsch, E. A., Kannan, R., Kleinberg, J. M., Papadimitriou, C. H., Raghavan, P., and Schöning, U. (2002).**
A deterministic $(2 - 2/(k + 1))^n$ algorithm for k -SAT based on local search.
Theoretical Computer Science, 289(1):69–83.
-  **Dantsin, E., Hirsch, E. A., and Wolpert, A. (2005).**
Clause shortening combined with pruning yields a new upper bound for deterministic SAT algorithms.
Electronic Colloquium on Computational Complexity (ECCC), 102.
-  **Davis, M., Logemann, G., and Loveland, D. (1962).**
A machine program for theorem proving.
Communications of the ACM, 5:394–397.





106 / 114

Bibliography VI

-  **Goerd, A. (1992).**
Davis-Putnam resolution versus unrestricted resolution.
Annals of Mathematics and Artificial Intelligence, 6:169–184.
-  **Gomes, C. P., Selman, B., Crato, N., and Kautz, H. (2000).**
Heavy-tailed phenomena in satisfiability and constraint satisfaction problems.
Journal of Automated Reasoning, 24(1–2):67–100.
-  **Grastien, A., Anbulagan, Rintanen, J., and Kelareva, E. (2007).**
Diagnosis of discrete-event systems using satisfiability algorithms.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 305–310. AAAI Press.
-  **Hirsch, E. A. (2000).**
New worst-case upper bounds for SAT.
Journal of Automated Reasoning, 24(4):397–420.
-  **Huang, J. (2007).**
The effect of restarts on the efficiency of clause learning.
In Veloso, M., editor, *Proceedings of the 20th International Joint Conference on Artificial Intelligence*, pages 2318–2323. AAAI Press / International Joint Conference on Artificial Intelligence.






108 / 114

Bibliography VII

-  Jackson, P. and Sheridan, D. (2005).
Clause form conversions for Boolean circuits.
In Hoos, H. H. and Mitchell, D. G., editors, *Theory and Applications of Satisfiability Testing, 7th International Conference, SAT 2004, Vancouver, BC, Canada, May 10-13, 2004, Revised Selected Papers*, volume 3542 of *Lecture Notes in Computer Science*, pages 183–198. Springer-Verlag.
-  Junttila, T. A. and Niemelä, I. (2000).
Towards an efficient tableau method for Boolean circuit satisfiability checking.
In Lloyd, J., Dahl, V., Furbach, U., Kerber, M., Lau, K.-K., Palamidessi, C., Pereira, L. M., Sagiv, Y., and Stuckey, P. J., editors, *Computational Logic - CL 2000; First International Conference London, UK, July 24-28, 2000 Proceedings*, number 1861 in *Lecture Notes in Computer Science*, pages 553–567. Springer-Verlag.
-  Kautz, H. and Selman, B. (1992).
Planning as satisfiability.
In Neumann, B., editor, *Proceedings of the 10th European Conference on Artificial Intelligence*, pages 359–363. John Wiley & Sons.
-  Kautz, H. and Selman, B. (1996).
Pushing the envelope: planning, propositional logic, and stochastic search.
In *Proceedings of the 13th National Conference on Artificial Intelligence and the 8th Innovative Applications of Artificial Intelligence Conference*, pages 1194–1201. AAAI Press.






109/114

Bibliography IX

-  Lynce, I. and Marques-Silva, J. (2006).
Efficient haplotype inference with boolean satisfiability.
In *Proceedings of the 21th National Conference on Artificial Intelligence (AAAI-2006)*, pages 104–109. AAAI Press.
-  Majercik, S. M. and Littman, M. L. (2003).
Contingent planning under uncertainty via stochastic satisfiability.
Artificial Intelligence, 147(1-2):119–162.
-  Manolios, P. and Vroon, D. (2007).
Efficient circuit to CNF conversion.
In Marques-Silva, J. and Sakallah, K. A., editors, *Proceedings of the 8th International Conference on Theory and Applications of Satisfiability Testing (SAT-2007)*, volume 4501 of *Lecture Notes in Computer Science*, pages 4–9. Springer-Verlag.
-  Mézard, M., Parisi, G., and Zecchina, R. (2002).
Analytic and algorithmic solution of random satisfiability problems.
Science, 297:812–815.
-  Mitchell, D., Selman, B., and Levesque, H. (1992).
Hard and easy distributions of SAT problems.
In Swartout, W., editor, *Proceedings of the 10th National Conference on Artificial Intelligence*, pages 459–465, San Jose, California. The MIT Press.

111/114

Bibliography VIII

-  Kleine Büning, H., Karpinski, M., and Flögel, A. (1995).
Resolution for quantified Boolean formulas.
Information and Computation, 117:12–18.
-  Larrabee, T. (1992).
Test pattern generation using boolean satisfiability.
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 11(1):4–15.
-  Latvala, T., Biere, A., Heljanko, K., and Junttila, T. (2004).
Simple bounded LTL model-checking.
In *Proceedings Intl. Conf. on Formal Methods in Computer-Aided Design, FMCAD'04*, pages 186–200.
-  Letz, R. (2002).
Lemma and model caching in decision procedures for quantified Boolean formula.
In Egly, U. and Fermüller, C. G., editors, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEUX 2002, Copenhagen, Denmark, July 30 - August 1, 2002, Proceedings*, volume 2381 of *Lecture Notes in Computer Science*, pages 160–175. Springer-Verlag.
-  Littman, M. L., Majercik, S. M., and Pitassi, T. (2001).
Stochastic Boolean satisfiability.
Journal of Automated Reasoning, 27:251–296.





110/114

Bibliography X


-  Moskewicz, M. W., Madigan, C. F., Zhao, Y., Zhang, L., and Malik, S. (2001).
Chaff: Engineering an Efficient SAT Solver.
In *Proceedings of the 38th ACM/IEEE Design Automation Conference (DAC'01)*, pages 530–535. ACM Press.
-  Rintanen, J. (1999a).
Constructing conditional plans by a theorem-prover.
Journal of Artificial Intelligence Research, 10:323–352.
-  Rintanen, J. (1999b).
Improvements to the evaluation of quantified Boolean formulae.
In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, pages 1192–1197. Morgan Kaufmann Publishers.
-  Rintanen, J. (2007).
Asymptotically optimal encodings of conformant planning in QBF.
In *Proceedings of the 22nd AAAI Conference on Artificial Intelligence (AAAI-07)*, pages 1045–1050. AAAI Press.
-  Rintanen, J., Heljanko, K., and Niemelä, I. (2006).
Planning as satisfiability: parallel plans and algorithms for plan search.
Artificial Intelligence, 170(12-13):1031–1080.

112/114

Bibliography XI

-  Selman, B., Kautz, H. A., and Cohen, B. (1994).
Noise strategies for improving local search.
In *Proceedings of the 19th National Conference on Artificial Intelligence (AAAI-2004) and the 16th Conference on Innovative Applications of Artificial Intelligence (IAAI-2004)*, pages 337–343. AAAI Press.
-  Streeter, M. and Smith, S. F. (2007).
Using decision procedures efficiently for optimization.
In Boddy, M., Fox, M., and Thiébaux, S., editors, *ICAPS 2007. Proceedings of the Seventeenth International Conference on Automated Planning and Scheduling*, pages 312–319.
-  Tseitin, G. S. (1962).
On the complexity of derivation in propositional calculus.
In Slisenko, A. O., editor, *Studies in Constructive Mathematics and Mathematical Logic, Part 2*, pages 115–125. Consultants Bureau, New York - London.
-  Wolfman, S. A. and Weld, D. S. (1999).
The LPSAT engine & its application to resource planning.
In Dean, T., editor, *Proceedings of the 16th International Joint Conference on Artificial Intelligence*, volume 1, pages 310–315. Morgan Kaufmann Publishers.

Bibliography XII

-  Zhang, L. and Malik, S. (2002).
Towards a symmetric treatment of satisfaction and conflicts in quantified Boolean formula evaluation.
In Van Hentenryck, P., editor, *Proceedings of the 8th International Conference on Principles and Practice of Constraint Programming*. Springer-Verlag.