Motivations

- ▷ Last ten years: impressive advance in boolean reasoning techniques (SAT)
  - extremely efficient solvers [96, 82, 14, 55, 61, 97]
  - hard "real-world" problems encoded into SAT (e.g.,
    - planning [52, 51, 30, 38],
    - model checking [19, 15, 1, 88, 94, 58, 23, 22, 20, 80],
    - circuit testing [85]
    - security & criptanalysis [57]
    - ...

Motivations (cont.)

- ▷ Recent years: using SAT solvers as boolean reasoning kernels for more expressive solvers
  - combine a SAT reasoner with a domain-specific solver
  - various domains:
    - Modal & description logics [41, 42, 48, 43, 37],
    - temporal reasoning [3],
    - resource planning [95],
    - verification of timed & hybrid systems [60, 6, 9, 84, 29, 65, 70, 8],
    - HW verification [21, 89, 29],
    - SW verification [21, 89],
    - reasoning in combined theories [62, 63, 81, 31, 7, 6, 12, 13, 89, 29, 56, 78, 90, 87, 86]

Content

PART 1:

PROPOSITIONAL SATISFIABILITY

Basics on SAT

Basic notation & definitions

- Boolean formula
  - $\top, \bot$ are formulas
  - A propositional atom $A_1, A_2, A_3, \ldots$ is a formula;
  - if φ_1 and φ_2 are formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ are formulas.
- Literal: a propositional atom A_i (positive literal) or its negation $\neg A_i$ (negative literal)
- N.B.: if $l := \neg A_i$, then $\neg l := A_i$
- $Atoms(\varphi)$: the set $\{A_1, \ldots, A_N\}$ of atoms occurring in φ.
- a boolean formula can be represented as a tree or as a DAG

Semantics of Boolean operators

φ_1	φ_2	$\neg\varphi_1$	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_1 \leftrightarrow \varphi_2$
$\bot$	$\bot$	$\top$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\bot$	$\top$	$\top$	$\top$	$\top$

N.B.:

$\varphi_1 \vee \varphi_2 := \neg(\neg\varphi_1 \wedge \neg\varphi_2)$,
$\varphi_1 \rightarrow \varphi_2 := (\neg\varphi_1 \vee \varphi_2)$,
$\varphi_1 \leftrightarrow \varphi_2 := (\varphi_1 \rightarrow \varphi_2) \wedge (\varphi_2 \rightarrow \varphi_1)$.

TREE and DAG representation of formulas: example

$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$

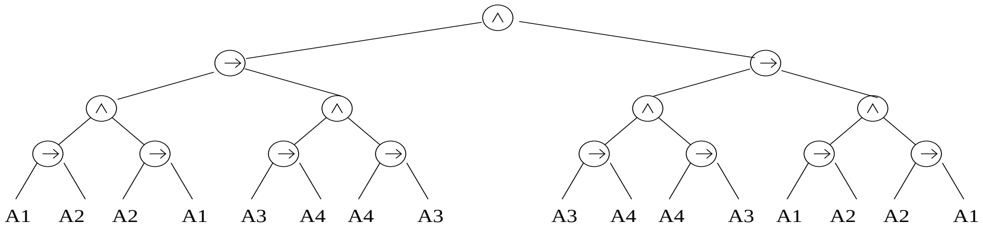
$$\Downarrow$$

$$(((A_1 \leftrightarrow A_2) \rightarrow (A_3 \leftrightarrow A_4)) \wedge$$
$$((A_3 \leftrightarrow A_4) \rightarrow (A_1 \leftrightarrow A_2)))$$

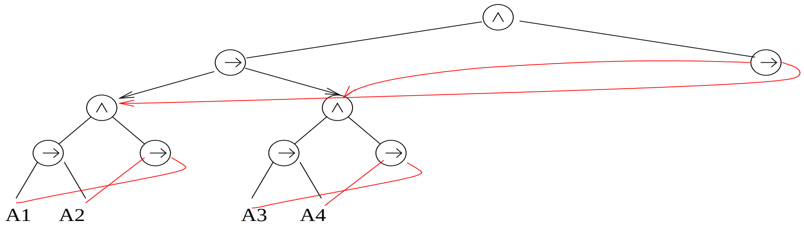
$$\Downarrow$$

$$(((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1)) \rightarrow ((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3))) \wedge$$
$$(((A_3 \rightarrow A_4) \wedge (A_4 \rightarrow A_3)) \rightarrow (((A_1 \rightarrow A_2) \wedge (A_2 \rightarrow A_1))))$$

TREE and DAG representation of formulas: example (cont)



Tree Representation



DAG Representation

Basic notation & definitions (cont)

- Total truth assignment μ for φ:
  $\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$.
- Partial Truth assignment μ for φ:
  $\mu : \mathcal{A} \longmapsto \{\top, \bot\}$, $\mathcal{A} \subset Atoms(\varphi)$.
- Set and formula representation of an assignment:
  - μ can be represented as a set of literals:
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies \{A_1, \neg A_2\}$
  - μ can be represented as a formula:
    EX: $\{\mu(A_1) := \top, \mu(A_2) := \bot\} \implies A_1 \wedge \neg A_2$

Basic notation & definitions (cont)

- $\mu \models \varphi$ (μ satisfies φ):
  - $\mu \models A_i \Longleftrightarrow \mu(A_i) = \top$
  - $\mu \models \neg\varphi \Longleftrightarrow$ *not* $\mu \models \varphi$
  - $\mu \models \varphi_1 \wedge \varphi_2 \Longleftrightarrow \mu \models \varphi_1$ *and* $\mu \models \varphi_2$
  - ...
- φ is satisfiable iff $\mu \models \varphi$ for some μ
- $\varphi_1 \models \varphi_2$ (φ_1 entails φ_2):
  $\varphi_1 \models \varphi_2$ iff for every μ $\mu \models \varphi_1 \Longrightarrow \mu \models \varphi_2$
- $\models \varphi$ (φ is valid):
  $\models \varphi$ iff for every μ $\mu \models \varphi$
- φ is valid $\Longleftrightarrow$ $\neg\varphi$ is not satisfiable

---

Equivalence and equi-satisfiability

- φ_1 and φ_2 are equivalent iff, for every μ,
  $\mu \models \varphi_1$ iff $\mu \models \varphi_2$
- φ_1 and φ_2 are equi-satisfiable iff
  exists μ_1 s.t. $\mu_1 \models \varphi_1$ iff exists μ_2 s.t. $\mu_2 \models \varphi_2$
- φ_1, φ_2 equivalent

  $$\Downarrow \quad \not\Uparrow$$

  φ_1, φ_2 equi-satisfiable

- EX: $\varphi_1 \vee \varphi_2$ and $(\varphi_1 \vee \neg A_3) \wedge (A_3 \vee \varphi_2)$, A_3 not in $\varphi_1 \vee \varphi_2$,
  are equi-satisfiable but not equivalent.

Complexity

- The problem of deciding the satisfiability of a propositional formula is NP-complete [24].
- The most important logical problems (validity, inference, entailment, equivalence, ...) can be straightforwardly reduced to satisfiability, and are thus (co)NP-complete.

$$\Downarrow$$

No existing worst-case-polynomial algorithm.

NNF, CNF and conversions

POLARITY of subformulas

Polarity: the number of nested negations modulo 2.

- Positive/negative occurrences
  - φ occurs positively in φ;
  - if $\neg\varphi_1$ occurs positively [negatively] in φ, then φ_1 occurs negatively [positively] in φ
  - if $\varphi_1 \wedge \varphi_2$ or $\varphi_1 \vee \varphi_2$ occur positively [negatively] in φ, then φ_1 and φ_2 occur positively [negatively] in φ;
  - if $\varphi_1 \rightarrow \varphi_2$ occurs positively [negatively] in φ, then φ_1 occurs negatively [positively] in φ and φ_2 occurs positively [negatively] in φ;
  - if $\varphi_1 \leftrightarrow \varphi_2$ occurs in φ, then φ_1 and φ_2 occur positively and negatively in φ;

Negative normal form (NNF)

- φ is in Negative normal form iff it is given only by applications of $\wedge, \vee$ to literals.
- every φ can be reduced into NNF:
  1. substituting all $\rightarrow$'s and $\leftrightarrow$'s:

$$\begin{array}{lcl} \varphi_1 \rightarrow \varphi_2 & \Longrightarrow & \neg\varphi_1 \vee \varphi_2 \\ \varphi_1 \leftrightarrow \varphi_2 & \Longrightarrow & (\neg\varphi_1 \vee \varphi_2) \wedge (\varphi_1 \vee \neg\varphi_2) \end{array}$$

  2. pushing down negations recursively:

$$\begin{array}{lcl} \neg(\varphi_1 \wedge \varphi_2) & \Longrightarrow & \neg\varphi_1 \vee \neg\varphi_2 \\ \neg(\varphi_1 \vee \varphi_2) & \Longrightarrow & \neg\varphi_1 \wedge \neg\varphi_2 \\ \neg\neg\varphi_1 & \Longrightarrow & \varphi_1 \end{array}$$

- The reduction is linear if a DAG representation is used.
- Preserves the equivalence of formulas.

NNF: example

$$(A_1 \leftrightarrow A_2) \leftrightarrow (A_3 \leftrightarrow A_4)$$

$$\Downarrow$$

$$((((A_1 \to A_2) \wedge (A_1 \leftarrow A_2)) \to ((A_3 \to A_4) \wedge (A_3 \leftarrow A_4))) \wedge$$
$$(((A_1 \to A_2) \wedge (A_1 \leftarrow A_2)) \leftarrow ((A_3 \to A_4) \wedge (A_3 \leftarrow A_4))))$$

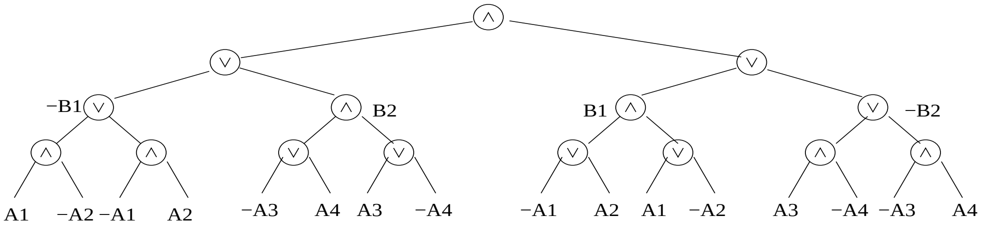
$$\Downarrow$$

$$((\neg((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee ((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4))) \wedge$$
$$(((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee \neg((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4))))$$

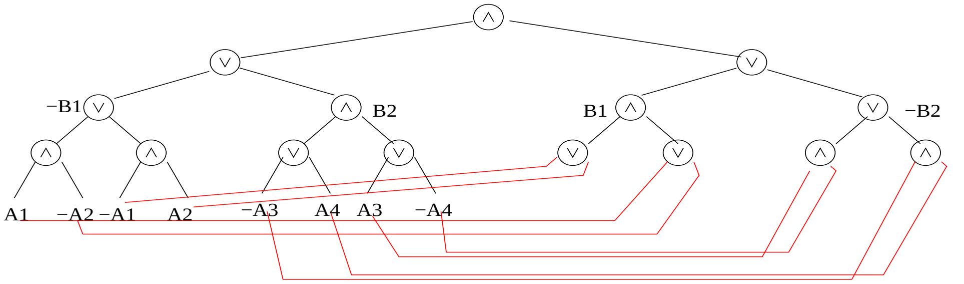
$$\Downarrow$$

$$((((A_1 \wedge \neg A_2) \vee (\neg A_1 \wedge A_2)) \vee ((\neg A_3 \vee A_4) \wedge (A_3 \vee \neg A_4))) \wedge$$
$$(((\neg A_1 \vee A_2) \wedge (A_1 \vee \neg A_2)) \vee ((A_3 \wedge \neg A_4) \vee (\neg A_3 \wedge A_4))))$$

NNF: example (cont)



Tree Representation



DAG Representation

N.B. For each non-literal subformula φ, φ and ¬φ have different representations ⟹ they are not shared.

Conjunctive Normal Form (CNF)

- φ is in Conjunctive normal form iff it is a conjunction of disjunctions of literals:

$$\bigwedge_{i=1}^{L} \bigvee_{j_i=1}^{K_i} l_{j_i}$$

- the disjunctions of literals $\bigvee_{j_i=1}^{K_i} l_{j_i}$ are called clauses
- Easier to handle: list of lists of literals.

  $\Longrightarrow$ no reasoning on the recursive structure of the formula

---

Classic CNF Conversion $CNF(\varphi)$

- Every φ can be reduced into CNF by, e.g.,
  1. converting it into NNF;
  2. applying recursively the DeMorgan's Rule:

$$(\varphi_1 \wedge \varphi_2) \vee \varphi_3 \quad \Longrightarrow \quad (\varphi_1 \vee \varphi_3) \wedge (\varphi_2 \vee \varphi_3)$$

- Worst-case exponential.
- $Atoms(CNF(\varphi)) = Atoms(\varphi)$.
- $CNF(\varphi)$ is equivalent to φ.
- Normal: if φ_1 equivalent to φ_2, then $CNF(\varphi_1)$ identical to $CNF(\varphi_2)$ modulo reordering.
- Rarely used in practice.

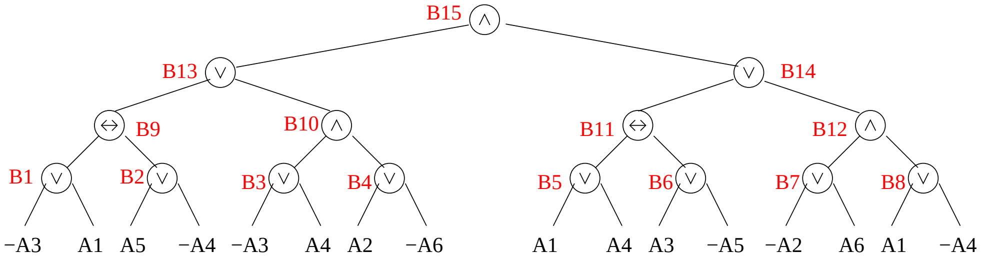
Labeling CNF conversion $CNF_{label}(\varphi)$ [71, 28]

- Every φ can be reduced into CNF by, e.g., applying recursively bottom-up the rules:

$$\varphi \implies \varphi[(l_i \vee l_j)|B] \wedge CNF(B \leftrightarrow (l_i \vee l_j))$$
$$\varphi \implies \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \leftrightarrow (l_i \wedge l_j))$$
$$\varphi \implies \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \leftrightarrow (l_i \leftrightarrow l_j))$$

  l_i, l_j being literals and B being a "new" variable.
- Worst-case linear.
- $Atoms(CNF_{label}(\varphi)) \supseteq Atoms(\varphi)$.
- $CNF_{label}(\varphi)$ is equi-satisfiable w.r.t. φ.
- Non-normal.
- More used in practice.

Labeling CNF conversion CNF_{label} – example



$CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1))$ $\wedge$
$\ldots$ $\wedge$
$CNF(B_8 \leftrightarrow (A_1 \vee \neg A_4))$ $\wedge$
$CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2))$ $\wedge$
$\ldots$ $\wedge$
$CNF(B_{12} \leftrightarrow (B_7 \wedge B_8))$ $\wedge$
$CNF(B_{13} \leftrightarrow (B_9 \vee B_{10}))$ $\wedge$
$CNF(B_{14} \leftrightarrow (B_{11} \vee B_{12}))$ $\wedge$
$CNF(B_{15} \leftrightarrow (B_{13} \wedge B_{14}))$ $\wedge$
B_{15}

Labeling CNF conversion CNF_{label} (improved)

- As in the previous case, applying instead the rules:

$$\varphi \Longrightarrow \varphi[(l_i \vee l_j)|B] \wedge CNF(B \rightarrow (l_i \vee l_j)) \ if \ (l_i \vee l_j) \ pos.$$

$$\varphi \Longrightarrow \varphi[(l_i \vee l_j)|B] \wedge CNF((l_i \vee l_j) \rightarrow B) \ if \ (l_i \vee l_j) \ neg.$$

$$\varphi \Longrightarrow \varphi[(l_i \wedge l_j)|B] \wedge CNF(B \rightarrow (l_i \wedge l_j)) \ if \ (l_i \wedge l_j) \ pos.$$

$$\varphi \Longrightarrow \varphi[(l_i \wedge l_j)|B] \wedge CNF((l_i \wedge l_j) \rightarrow B) \ if \ (l_i \wedge l_j) \ neg.$$

$$\varphi \Longrightarrow \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF(B \rightarrow (l_i \leftrightarrow l_j)) \ if \ (l_i \leftrightarrow l_j) \ pos.$$

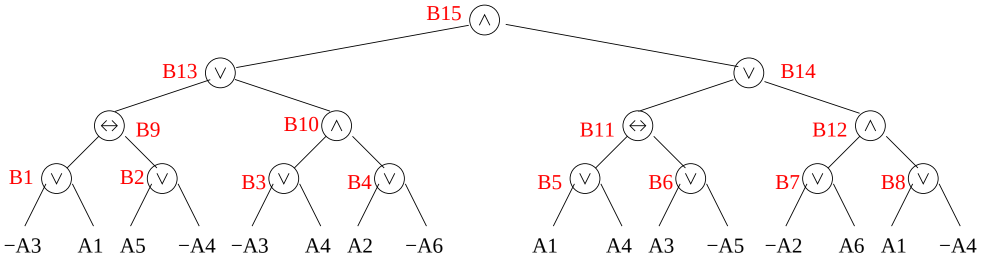
$$\varphi \Longrightarrow \varphi[(l_i \leftrightarrow l_j)|B] \wedge CNF((l_i \leftrightarrow l_j) \rightarrow B) \ if \ (l_i \leftrightarrow l_j) \ neg.$$

- Smaller in size:

$$CNF(B \rightarrow (l_i \vee l_j)) = (\neg B \vee l_i \vee l_j)$$

$$CNF(((l_i \vee l_j) \rightarrow B)) = (\neg l_i \vee B) \wedge (\neg l_j \vee B)$$

Labeling CNF conversion CNF_{label} – example



Basic		Improved	
$CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1))$	$\wedge$	$CNF(B_1 \leftrightarrow (\neg A_3 \vee A_1))$	$\wedge$
$\ldots$	$\wedge$	$\ldots$	$\wedge$
$CNF(B_8 \leftrightarrow (A_1 \vee \neg A_4))$	$\wedge$	$CNF(B_8 \rightarrow (A_1 \vee \neg A_4))$	$\wedge$
$CNF(B_9 \leftrightarrow (B_1 \leftrightarrow B_2))$	$\wedge$	$CNF(B_9 \rightarrow (B_1 \leftrightarrow B_2))$	$\wedge$
$\ldots$	$\wedge$	$\ldots$	$\wedge$
$CNF(B_{12} \leftrightarrow (B_7 \wedge B_8))$	$\wedge$	$CNF(B_{12} \rightarrow (B_7 \wedge B_8))$	$\wedge$
$CNF(B_{13} \leftrightarrow (B_9 \vee B_{10}))$	$\wedge$	$CNF(B_{13} \rightarrow (B_9 \vee B_{10}))$	$\wedge$
$CNF(B_{14} \leftrightarrow (B_{11} \vee B_{12}))$	$\wedge$	$CNF(B_{14} \rightarrow (B_{11} \vee B_{12}))$	$\wedge$
$CNF(B_{15} \leftrightarrow (B_{13} \wedge B_{14}))$	$\wedge$	$CNF(B_{15} \rightarrow (B_{13} \wedge B_{14}))$	$\wedge$
B_{15}		B_{15}	

Basic SAT techniques

Truth Tables

- Exhaustive evaluation of all subformulas:

φ_1	φ_2	$\varphi_1 \wedge \varphi_2$	$\varphi_1 \vee \varphi_2$	$\varphi_1 \rightarrow \varphi_2$	$\varphi_1 \leftrightarrow \varphi_2$
$\bot$	$\bot$	$\bot$	$\bot$	$\top$	$\top$
$\bot$	$\top$	$\bot$	$\top$	$\top$	$\bot$
$\top$	$\bot$	$\bot$	$\top$	$\bot$	$\bot$
$\top$	$\top$	$\top$	$\top$	$\top$	$\top$

- Requires polynomial space.
- Never used in practice.

Semantic tableaux [83]

- Search for an assignment satisfying φ
- applies recursively elimination rules to the connectives
- If a branch contains A_i and $\neg A_i$, (ψ_i and $\neg\psi_1$) for some i, the branch is closed, otherwise it is open.
- if no rule can be applied to an open branch μ, then $\mu \models \varphi$;
- if all branches are closed, the formula is not satisfiable;

Tableau elimination rules

$$\frac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}} \qquad \frac{\neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\neg\varphi_1\\ \neg\varphi_2\end{array}} \qquad \frac{\neg(\varphi_1 \rightarrow \varphi_2)}{\begin{array}{c}\varphi_1\\ \neg\varphi_2\end{array}} \qquad (\wedge\text{-elimination})$$

$$\frac{\neg\neg\varphi}{\varphi} \qquad (\neg\neg\text{-elimination})$$

$$\frac{\varphi_1 \vee \varphi_2}{\varphi_1 \quad \varphi_2} \qquad \frac{\neg(\varphi_1 \wedge \varphi_2)}{\neg\varphi_1 \quad \neg\varphi_2} \qquad \frac{\varphi_1 \rightarrow \varphi_2}{\neg\varphi_1 \quad \varphi_2} \qquad (\vee\text{-elimination})$$

$$\frac{\varphi_1 \leftrightarrow \varphi_2}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \varphi_2 & \neg\varphi_2\end{array}} \qquad \frac{\neg(\varphi_1 \leftrightarrow \varphi_2)}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \neg\varphi_2 & \varphi_2\end{array}} \qquad (\leftrightarrow\text{-elimination}).$$

Semantic Tableaux – example

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$

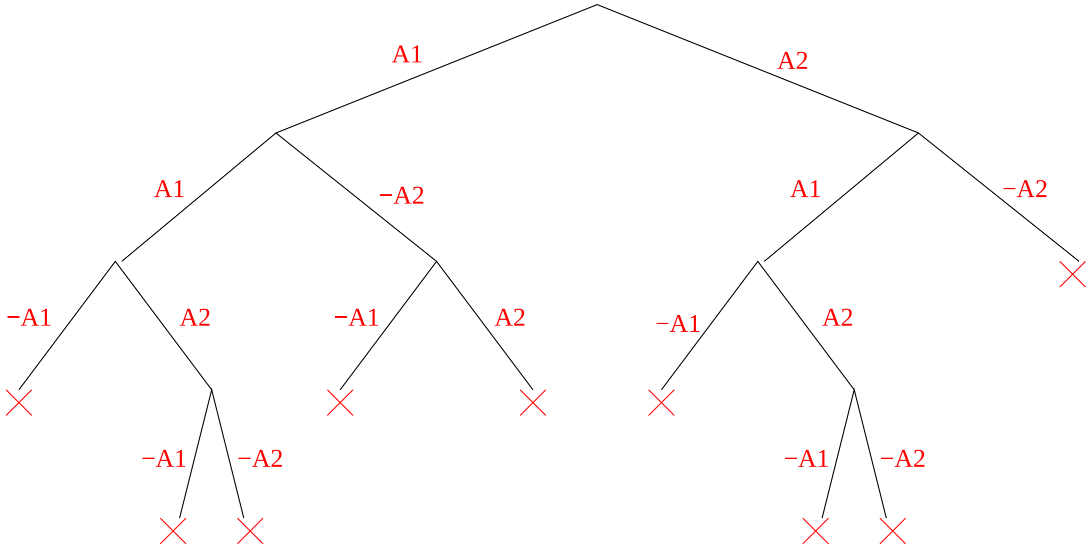


Tableau algorithm

```
function Tableau(Γ)
    if $A_i \in \Gamma$ and $\neg A_i \in \Gamma$                          /* branch closed */
        then return False;
    if $(\varphi_1 \wedge \varphi_2) \in \Gamma$                            /* ∧-elimination */
        then return Tableau($\Gamma \cup \{\varphi_1, \varphi_2\} \setminus \{(\varphi_1 \wedge \varphi_2)\}$);
    if $(\neg\neg\varphi_1) \in \Gamma$                                     /* ¬¬-elimination */
        then return Tableau($\Gamma \cup \{\varphi_1\} \setminus \{(\neg\neg\varphi_1)\}$);
    if $(\varphi_1 \vee \varphi_2) \in \Gamma$                              /* ∨-elimination */
        then return    Tableau($\Gamma \cup \{\varphi_1\} \setminus \{(\varphi_1 \vee \varphi_2)\}$) or
                       Tableau($\Gamma \cup \{\varphi_2\} \setminus \{(\varphi_1 \vee \varphi_2)\}$);
    ...
    return True;                                                            /* branch expanded */
```

Semantic Tableaux – summary

- Handles all propositional formulas (CNF not required).
- Branches on disjunctions
- Intuitive, modular, easy to extend

  $\Longrightarrow$ loved by logicians.
- Rather inefficient

  $\Longrightarrow$ avoided by computer scientists.
- Requires polynomial space

DPLL [27, 26]

- Davis-Putnam-Longeman-Loveland procedure (DPLL)
- Tries to build recursively an assignment μ satisfying φ;
- At each recursive step assigns a truth value to (all instances of) one atom.
- Performs deterministic choices first.

DPLL rules

$$\frac{\varphi_1 \wedge (l)}{\varphi_1[l|\top]} \ (Unit)$$

$$\frac{\varphi}{\varphi[l|\top]} \ (l\ Pure)$$

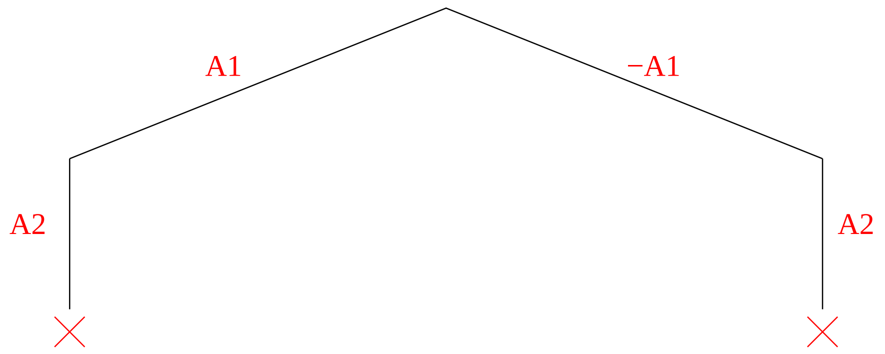
$$\frac{\varphi}{\varphi[l|\top] \quad \varphi[l|\bot]} \ (split)$$

(l is a pure literal in φ iff it occurs only positively).

- Split applied if and only if the others cannot be applied.
- Equivalent formalism described in [90]

DPLL – example

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$



DPLL Algorithm

```
function DPLL($\varphi, \mu$)
    if $\varphi = \top$                                    /* base      */
        then return True;
    if $\varphi = \bot$                                    /* backtrack */
        then return False;
    if {a unit clause $(l)$ occurs in $\varphi$}           /* unit      */
        then return DPLL($assign(l, \varphi), \mu \wedge l$);
    if {a literal $l$ occurs pure in $\varphi$}            /* pure      */
        then return DPLL($assign(l, \varphi), \mu \wedge l$);
    l := choose-literal($\varphi$);                        /* split     */
    return  DPLL($assign(l, \varphi), \mu \wedge l$)  or
            DPLL($assign(\neg l, \varphi), \mu \wedge \neg l$);
```

DPLL – summary

- Handles CNF formulas (non-CNF variant known [5, 40]).
- Branches on truth values
  $\Longrightarrow$all instances of an atom assigned simultaneously
- Postpones branching as much as possible.
- Mostly ignored by logicians.
- Probably the most efficient SAT algorithm
  $\Longrightarrow$ loved by computer scientists.
- Requires polynomial space
- Choose_literal() critical!
- Many very efficient implementations [96, 82, 14, 61].
- A library: SIM [39]

Stalmark's procedure [79]

– Using triplets to represent formulas (represents DAGS)

$$\underbrace{(A_1 \wedge \overbrace{(A_2 \wedge A_3)}^{B_3})}_{B_1} \vee \underbrace{(\neg A_1 \wedge \neg \overbrace{(A_2 \wedge A_3)}^{B_3})}_{B_2}$$

$$\Downarrow$$

$$\begin{gathered}(B_1 \vee B_2) \wedge \\ (B_1 \leftrightarrow A_1 \wedge B_3) \wedge \\ (B_2 \leftrightarrow \neg A_1 \wedge \neg B_3) \wedge \\ (B_3 \leftrightarrow A_2 \wedge A_3)\end{gathered}$$

– Breadth first search up to depth 2

Stalmark's procedure (cont.)

– Try both sides of a branch to find forced decisions. EX:

- $(A \vee B) \wedge (\neg A \vee C) \wedge (\neg A \vee B) \wedge (A \vee D)$
-

$$\begin{array}{ccc} A = \bot & \Longrightarrow & B = \top, D = \top \\ A = \top & \Longrightarrow & B = \top, C = \top \end{array}$$

$$\Downarrow$$

$$B = \top$$

– Repeat for all variables (depth 1) and variable pairs (depth 2)

– if not sufficient, run a DPLL-like procedure on the resulting formula

Stalmark's procedure – summary

- Handles non-CNF formulas in DAG form
- Branches on truth values (of subformulas)
- very efficient with particular kinds of formulas (e.g., circuits)
- Requires polynomial space
- No freely available implementation

Ordered Binary Decision Diagrams (OBDDs) [19]

- Normal representation of a boolean formula.
- "If-then-else" binary DAGs with two leaves: 1 and 0
- Variable ordering $A_1, A_2, ..., A_n$ imposed a priory.
- Paths leading to 1 represent models

  Paths leading to 0 represent counter-models
- Once built, logical operations (satisfiability, validity, equivalence) immediate.
- Finds all models.

(Implicit) OBDD structure

- $OBDD(\top, \{...\}) = 1$,
- $OBDD(\bot, \{...\}) = 0$,
- $OBDD(\varphi, \{A_1, A_2, ..., A_n\}) =$
  $if\ A_1$
  $then\ OBDD(\varphi[A_1|\top], \{A_2, ..., A_n\})$
  $else\ OBDD(\varphi[A_1|\bot], \{A_2, ..., A_n\})$

OBDD - Examples

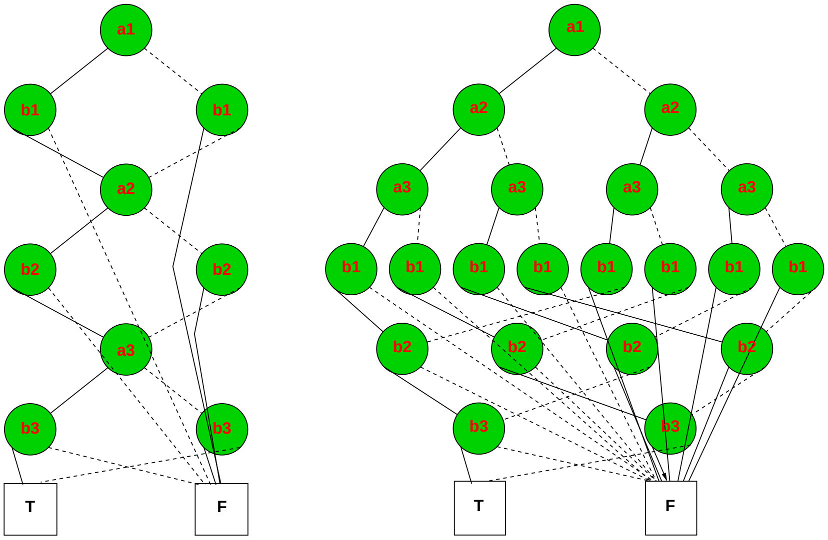


Figure 1: OBDDs of $(a_1 \leftrightarrow b_1) \wedge (a_2 \leftrightarrow b_2) \wedge (a_3 \leftrightarrow b_3)$ with different variable orderings

Incrementally building an OBDD

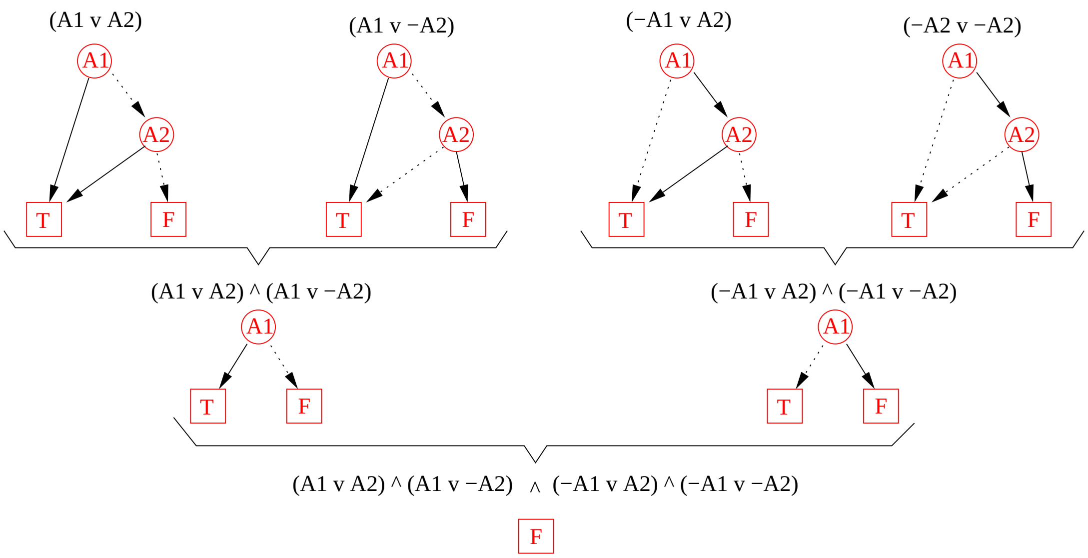
- $obdd\_build(\top, \{...\}) := 1$,
- $obdd\_build(\bot, \{...\}) := 0$,
- $obdd\_build((\varphi_1\ op\ \varphi_2), \{A_1, ..., A_n\}) :=$

$$
\begin{array}{ll}
obdd\_merge(& op, \\
 & obdd\_build(\varphi_1, \{A_1, ..., A_n\}), \\
 & obdd\_build(\varphi_2, \{A_1, ..., A_n\}), \\
 & \{A_1, ..., A_n\} \\
)
\end{array}
$$

$op \in \{\wedge, \vee, \rightarrow, \leftrightarrow\}$

OBBD incremental building – example

$$\varphi = (A_1 \vee A_2) \wedge (A_1 \vee \neg A_2) \wedge (\neg A_1 \vee A_2) \wedge (\neg A_1 \vee \neg A_2)$$



OBDD – summary

- Handle all propositional formulas (CNF not required).
- (Implicitly) branch on truth values.
- Find all models.
- Factorize common parts of the search tree (DAG)
- Require setting a variable ordering a priori (critical!)
  $\Longrightarrow$ cannot postpone branching
- Very efficient for some problems (circuits, model checking).
- Require exponential space in worst-case
- Used by Hardware community, ignored by logicians, recently introduced in computer science.

Incomplete SAT techniques: GSAT, WSAT [76, 75]

- Hill-Climbing techniques: GSAT, WSAT
- looks for a complete assignment;
- starts from a random assignment;
- Greedy search: looks for a better “neighbor” assignment
- Avoid local minima: restart & random walk

GSAT algorithm

```
function GSAT(φ)
    for $i := 1$ to Max-tries do
        $\mu$ := rand-assign(φ);
        for $j := 1$ to Max-flips do
            if ($score(\varphi, \mu) = 0$)
                then return True;
                else Best-flips := hill-climb($\varphi, \mu$);
                    $A_i$ := rand-pick(Best-flips);
                    $\mu$ := flip($A_i, \mu$);
        end
    end
    return "no satisfying assignment found".
```

GSAT & WSAT– summary

- Handle only CNF formulas.
- Incomplete
- Extremely efficient for some (satisfiable) problems.
- Require polynomial space
- Used in Artificial Intelligence (e.g., planning)
- Non-CNF Variants: NC-GSAT [73], DAG-SAT [74]

DPLL Heuristics & Optimizations

Techniques to achieve efficiency in DPLL

- Preprocessing: preprocess the input formula so that to make it easier to solve
- Look-ahead: exploit information about the remaining search space
  - unit propagation
  - pure literal
  - forward checking (splitting heuristics)
- Look-back: exploit information about search which has already taken place
  - Backjumping
  - Learning

Variants of DPLL

DPLL is a family of algorithms.

- different splitting heuristics
- preprocessing: (subsumption, 2-simplification)
- backjumping
- learning
- random restart
- horn relaxation
- ...

Iterative description of DPLL [82, 97]

```
status = preprocess();
if (status!=UNKNOWN) return status;
while(1) {
    decide_next_branch();
    while (1) {
        status = deduce();
        if (status == CONFLICT) {
            blevel = analyze_conflict();
            if (blevel == 0)
                return UNSATISFIABLE; else
         backtrack(blevel);
        }
        else if (status == SATISFIABLE)
            return SATISFIABLE;
        else break;
}   }
```

Splitting heuristics - Choose_literal()

- Split is the source of non-determinism for DPLL
- Choose_literal() critical for efficiency
- many split heuristics conceived in literature.

Some example heuristics

- MOMS heuristics: pick the literal occurring most often in the minimal size clauses

  ⟹fast and simple
- Jeroslow-Wang: choose the literal with maximum

  $$score(l) := \Sigma_{l \in c \ \& \ c \in \varphi} \ 2^{-|c|}$$

  ⟹estimates l's contribution to the satisfiability of φ
- Satz [55]: selects a candidate set of literals, perform unit propagation, chooses the one leading to smaller clause set

  ⟹maximizes the effects of unit propagation
- Chaff's VSIDS [61]: variable state independent decaying sum
- ...

Some preprocessing techniques

- Sorting+subsumption:

$$\varphi_1 \wedge (l_2 \vee l_1) \wedge \varphi_2 \wedge (l_2 \vee l_3 \vee l_1) \wedge \varphi_3$$

$$\Downarrow$$

$$\varphi_1 \wedge (l_1 \vee l_2) \wedge \varphi_2 \wedge \varphi_3$$

Some preprocessing techniques (cont.)

- 2-simplify [17]: exploiting binary clauses.
- **Repeat:**
  1. build the implication graph induced by literals
  2. detect strongly connected cycles
     $\Longrightarrow$equivalence classes of literals
  3. perform substitutions
  4. perform unit and pure.

  Until no more simplification possible.
- Very useful for many application domains.
- Improvement: Hypre [11].1

Conflict-directed backtracking (backjumping) [14, 82]

- Idea: when a branch fails,
  1. reveal the sub-assignment causing the failure (conflict set)
  2. backtrack to the most recent branching point in the conflict set
- a conflict set is constructed from the conflict clause by tracking backwards the unit-implications causing it and by keeping the branching literals.
- when a branching point fails, a conflict set is obtained by resolving the two conflict sets of the two branches.
- may avoid lots of redundant search.

Conflict-directed backtracking – example

$\neg A_1 \vee A_2$

$\neg A_1 \vee A_3 \vee A_9$

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$

$A_1 \vee A_8$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$

$\neg A_1 \vee A_3 \vee A_9$

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$

$A_1 \vee A_8$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots\}$ (initial assignment)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$

$\neg A_1 \vee A_3 \vee A_9$

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$ $\quad$ $true \Longrightarrow removed$

$A_1 \vee A_8$ $\quad$ $true \Longrightarrow removed$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1\}$ (branch on A_1)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_1 \vee A_3 \vee A_9$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$ $\quad$ *true* $\Longrightarrow$ *removed*

$A_1 \vee A_8$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3\}$
(unit A_2, A_3)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_1 \vee A_3 \vee A_9$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_2 \vee \neg A_3 \vee A_4$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$ $\quad$ *true* $\Longrightarrow$ *removed*

$A_1 \vee A_8$ $\quad$ *true* $\Longrightarrow$ *removed*

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3, A_4\}$
(unit A_4)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$	$true \Longrightarrow removed$
$\neg A_1 \vee A_3 \vee A_9$	$true \Longrightarrow removed$
$\neg A_2 \vee \neg A_3 \vee A_4$	$true \Longrightarrow removed$
$\neg A_4 \vee A_5 \vee A_{10}$	$true \Longrightarrow removed$
$\neg A_4 \vee A_6 \vee A_{11}$	$true \Longrightarrow removed$
$\neg A_5 \vee \neg A_6$	$false \Longrightarrow conflict$
$A_1 \vee A_7 \vee \neg A_{12}$	$true \Longrightarrow removed$
$A_1 \vee A_8$	$true \Longrightarrow removed$
$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$	
$\ldots$	

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, A_1, A_2, A_3, A_4, A_5, A_6\}$
(unit A_5, A_6)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2$	$true \Longrightarrow removed$
$\neg A_1 \vee A_3 \vee A_9$	$true \Longrightarrow removed$
$\neg A_2 \vee \neg A_3 \vee A_4$	$true \Longrightarrow removed$
$\neg A_4 \vee A_5 \vee A_{10}$	$true \Longrightarrow removed$
$\neg A_4 \vee A_6 \vee A_{11}$	$true \Longrightarrow removed$
$\neg A_5 \vee \neg A_6$	$false \Longrightarrow conflict$
$A_1 \vee A_7 \vee \neg A_{12}$	$true \Longrightarrow removed$
$A_1 \vee A_8$	$true \Longrightarrow removed$
$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$	
$\ldots$	

$\Longrightarrow$Conflict set: $\{\neg A_9, \neg A_{10}, \neg A_{11}, A_1\}$ $\Longrightarrow$backtrack to A_1

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2 \qquad true \Longrightarrow removed$

$\neg A_1 \vee A_3 \vee A_9 \qquad true \Longrightarrow removed$

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12}$

$A_1 \vee A_8$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1\}$ (branch on $\neg A_1$)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \vee A_2 \qquad true \Longrightarrow removed$

$\neg A_1 \vee A_3 \vee A_9 \qquad true \Longrightarrow removed$

$\neg A_2 \vee \neg A_3 \vee A_4$

$\neg A_4 \vee A_5 \vee A_{10}$

$\neg A_4 \vee A_6 \vee A_{11}$

$\neg A_5 \vee \neg A_6$

$A_1 \vee A_7 \vee \neg A_{12} \qquad true \Longrightarrow removed$

$A_1 \vee A_8 \qquad true \Longrightarrow removed$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13} \; false \Longrightarrow conflict$

$\ldots$

$\{\ldots, \neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}, \ldots, \neg A_1, A_7, A_8\}$

(unit A_7, A_8)

Conflict-directed backtracking – example (cont.)

$\neg A_1 \lor A_2 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_1 \lor A_3 \lor A_9 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_2 \lor \neg A_3 \lor A_4$

$\neg A_4 \lor A_5 \lor A_{10}$

$\neg A_4 \lor A_6 \lor A_{11}$

$\neg A_5 \lor \neg A_6$

$A_1 \lor A_7 \lor \neg A_{12} \quad \mathit{true} \Longrightarrow \mathit{removed}$

$A_1 \lor A_8 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_7 \lor \neg A_8 \lor \neg A_{13} \quad \mathit{false} \Longrightarrow \mathit{conflict}$

$\ldots$

$\Longrightarrow$conflict set: $\{A_{12}, A_{13}, \neg A_1\}$.

Conflict-directed backtracking – example (cont.)

$\neg A_1 \lor A_2 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_1 \lor A_3 \lor A_9 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_2 \lor \neg A_3 \lor A_4$

$\neg A_4 \lor A_5 \lor A_{10}$

$\neg A_4 \lor A_6 \lor A_{11}$

$\neg A_5 \lor \neg A_6$

$A_1 \lor A_7 \lor \neg A_{12} \quad \mathit{true} \Longrightarrow \mathit{removed}$

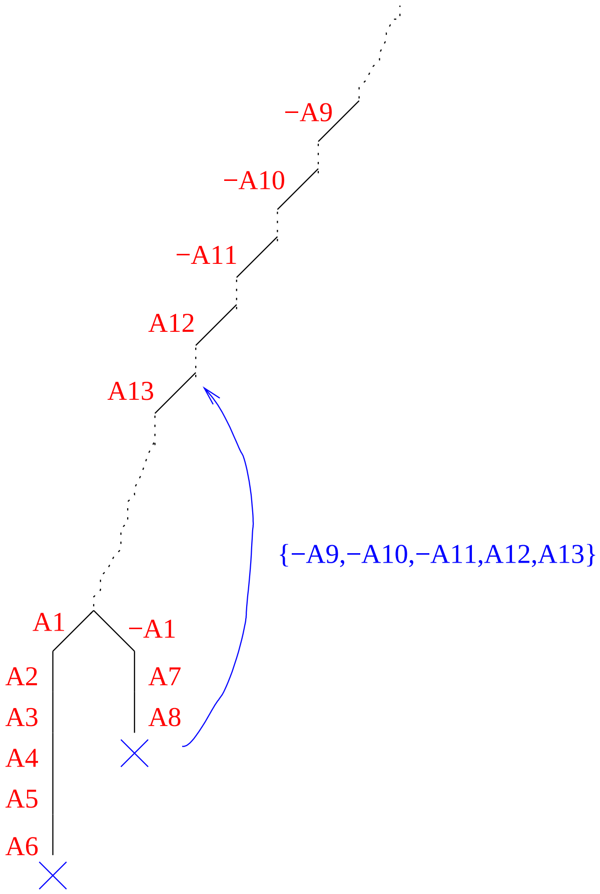
$A_1 \lor A_8 \quad \mathit{true} \Longrightarrow \mathit{removed}$

$\neg A_7 \lor \neg A_8 \lor \neg A_{13} \quad \mathit{false} \Longrightarrow \mathit{conflict}$

$\ldots$

$\Longrightarrow$conflict set: $\{A_{12}, A_{13}, \neg A_1\} \ldots \lor \{\neg A_9, \neg A_{10}, \neg A_{11}, A_1\}$

$\Longrightarrow \{\neg A_9, \neg A_{10}, \neg A_{11}, A_{12}, A_{13}\} \Longrightarrow$ backtrack to A_{13}.



Learning [14, 82]

- Idea: When a conflict set C is revealed, then $\neg C$ can be added to the clause set
  $\Longrightarrow$DPLL will never again generate an assignment containing C.
- May avoid a lot of redundant search.
- Problem: may cause a blowup in space
  $\Longrightarrow$techniques to control learning and to drop learned clauses when necessary

Learning – example (cont.)

$\neg A_1 \vee A_2 \quad true \Longrightarrow removed$

$\neg A_1 \vee A_3 \vee A_9 \quad true \Longrightarrow removed$

$\neg A_2 \vee \neg A_3 \vee A_4 \quad true \Longrightarrow removed$

$\neg A_4 \vee A_5 \vee A_{10} \quad true \Longrightarrow removed$

$\neg A_4 \vee A_6 \vee A_{11} \quad true \Longrightarrow removed$

$\neg A_5 \vee \neg A_6 \quad false \Longrightarrow conflict$

$A_1 \vee A_7 \vee \neg A_{12} \quad true \Longrightarrow removed$

$A_1 \vee A_8 \quad true \Longrightarrow removed$

$\neg A_7 \vee \neg A_8 \vee \neg A_{13}$

$\ldots$

$A_9 \vee A_{10} \vee A_{11} \vee \neg A_1 \quad learned\ clause$

$\Longrightarrow$Conflict set: $\{\neg A_9, \neg A_{10}, \neg A_{11}, A_1\}$

$\Longrightarrow$learn $A_9 \vee A_{10} \vee A_{11} \vee \neg A_1$

PART 2:

BEYOND PROPOSITIONAL SATISFIABILITY

Goal

Integrate SAT procedures with domain-specific solvers in an efficient way

Different viewpoints:

- (Logicians' communities) Provide a new "SAT based" general framework from which to build efficient decision procedures (alternative, e.g., to semantic tableaux)
- (SAT community) Extend SAT techniques to more expressive domains (preserving efficiency)
- (Decision procedures community) Optimize the boolean component of reasoning

Key issues

- Correctness, completeness & termination
  - A general logic framework
  - A general integration schema
- Efficiency
  - Efficiency issues of the SAT procedure
  - Efficiency issues of the domain-specific solver
  - Efficiency of the integration

    $\Longrightarrow$A procedure integrating a very efficient SAT solver with a very efficient domain-specific solver may be dramatically inefficient if the integration is not done properly.

Formal Framework

Ingredients

– A logic language $\mathcal{L}$ extending boolean logic:
  - Language-specific atomic expression are formulas (e.g., $P(x)$, $\Box(A_1 \vee \Box A_2)$, $(x - y \geq 6)$, $\exists\ \textsc{Children}\ (\textsc{male} \wedge \textsc{teen})$)
  - if φ_1 and φ_2 formulas, then $\neg\varphi_1$, $\varphi_1 \wedge \varphi_2$, $\varphi_1 \vee \varphi_2$, $\varphi_1 \rightarrow \varphi_2$, $\varphi_1 \leftrightarrow \varphi_2$ are formulas.
  - Nothing else is a formula (e.g., no external quantifiers!)

Ingredients (cont.)

– A semantic for $\mathcal{L}$ extending standard boolean one:

$$\begin{array}{lcl}
M \models \psi, (\psi\ atomic) & \Longleftrightarrow & \text{[definition specific for } \mathcal{L}\text{]} \\
M \models \neg\phi & \Longleftrightarrow & M \not\models \phi \\
M \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & M \models \varphi_1 \textbf{ and } M \models \varphi_2 \\
M \models \varphi_1 \vee \varphi_2 & \Longleftrightarrow & M \models \varphi_1 \textbf{ or } M \models \varphi_2 \\
M \models \varphi_1 \rightarrow \varphi_2 & \Longleftrightarrow & \textbf{if}M \models \varphi_1 \textbf{ then } M \models \varphi_2 \\
M \models \varphi_1 \leftrightarrow \varphi_2 & \Longleftrightarrow & M \models \varphi_1 \textbf{ iff } M \models \varphi_2
\end{array}$$

Ingredients (cont.)

– A language-specific procedure $\mathcal{L}$-SOLVE able to decide the satisfiability of lists of atomic expressions and their negations E.g.:

- FO-SOLVE($\{P(x,a), P(b,y)\}$) $\Longrightarrow$Sat
- K-SOLVE($\{\Box(A_1 \rightarrow A_2), \Box(A_1), \neg\Box(A_2)\}$) $\Longrightarrow$Unsat
- MATH-SOLVE($\{(x-y \leq 3), (y-z \leq 4), \neg(x-z \leq 8)\}$) $\Longrightarrow$Unsat
- $\mathcal{ALC}$-SOLVE $$\left(\left\{\begin{array}{l} \forall\,\textsc{Children}\ (\neg\textsc{male} \vee \textsc{teen}), \\ \forall\,\textsc{Children}\ (\textsc{male}), \\ \exists\,\textsc{Children}\ (\neg\ \textsc{teen}) \end{array}\right\}\right)$$ $\Longrightarrow$Unsat

Definitions: atoms, literals

- An atom is every formula in $\mathcal{L}$ whose main connective is not a boolean operator.
- A literal is either an atom (a positive literal) or its negation (a negative literal).
- Examples:
  $P(x),\ \neg\forall x.Q(x, f(a))$
  $\Box(A_1 \vee \Box A_2),\ \neg\Box(A_2 \rightarrow \Box(A_3 \vee A_4))$
  $(x - y \geq 6),\ \neg(z - y < 2),$
  $\exists\ \textsc{Children}\ (\textsc{male} \wedge \textsc{teen}), \neg\forall\ \textsc{Parent}\ (\textsc{old})$
- $Atoms(\varphi)$: the set of top-level atoms in φ.

Definitions: total truth assignment

- We call a total truth assignment μ for φ a total function

$$\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$$

- We represent a total truth assignment μ either as a set of literals

$$\mu = \{\alpha_1, \ldots, \alpha_N, \neg\beta_1, \ldots, \neg\beta_M, A_1, \ldots, A_R, \neg A_{R+1}, \ldots, \neg A_S\},$$

or as a boolean formula

$$\mu = \bigwedge_i \alpha_i \wedge \bigwedge_j \neg\beta_j \wedge \bigwedge_{k=1}^{R} A_k \wedge \bigwedge_{h=R+1}^{S} \neg A_h$$

Definitions: partial truth assignment

- We call a partial truth assignment μ for φ a partial function

$$\mu : Atoms(\varphi) \longmapsto \{\top, \bot\}$$

- Partial truth assignments can be represented as sets of literals or as boolean functions, as before.
- A partial truth assignment μ for φ is a subset of a total truth assignment for φ.
- If $\mu_2 \subseteq \mu_1$, then we say that μ_1 extends μ_2 and that μ_2 subsumes μ_1.
- a conflict set for μ_1 is an inconsistent subset $\mu_2 \subseteq \mu_1$ s.t. no strict subset of μ_2 is inconsistent.

Definitions: total and partial truth assignment (cont.)

Remark:

- Syntactically identical instances of the same atom in φ are always assigned identical truth values.
  E.g., $... \wedge ((t_1 \geq t_2) \vee A_1) \wedge ((t_1 \geq t_2) \vee A_2) \wedge ...$
- Equivalent but syntactically different atoms in φ may (in principle) be assigned different truth values.
  E.g., $... \wedge ((t_1 \geq t_2) \vee A_1) \wedge ((t_2 \leq t_1) \vee A_2) \wedge ...$

Definition: propositional satisfiability in $\mathcal{L}$

A truth assignment μ for φ propositionally satisfies φ in $\mathcal{L}$, written $\mu \models_p \varphi$, iff it makes φ evaluate to $\top$:

$$\begin{array}{lll}
\mu \models_p \varphi_1,\ \varphi_1 \in Atoms(\varphi) & \Longleftrightarrow & \varphi_1 \in \mu; \\
\mu \models_p \neg\varphi_1 & \Longleftrightarrow & \mu \not\models_p \varphi_1; \\
\mu \models_p \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & \mu \models_p \varphi_1 \textit{ and } \mu \models_p \varphi_2. \\
\ldots & \ldots & \ldots
\end{array}$$

– A partial assignment μ propositionally satisfies φ iff all total assignments extending μ propositionally satisfy φ.

Definition: propositional satisfiability in $\mathcal{L}$ (cont)

– Intuition: If φ is seen as a boolean combination of its atoms, $\models_p$ is standard propositional satisfiability.

– Atoms seen as (recognizable) blackboxes

– The definitions of $\varphi_1 \models_p \varphi_2$, $\models_p \varphi$ is straightforward.

– $\models_p$ stronger than $\models$: if $\varphi_1 \models_p \varphi_2$, then $\varphi_1 \models \varphi_2$, but not vice versa.
E.g., $(v_1 \le v_2) \wedge (v_2 \le v_3) \models (v_1 \le v_3)$, but
$(v_1 \le v_2) \wedge (v_2 \le v_3) \not\models_p (v_1 \le v_3)$.

Satisfiability and propositional satisfiability in $\mathcal{L}$

Proposition: φ is satisfiable in $\mathcal{L}$ iff there exists a truth assignment μ for φ s.t.

- $\mu \models_p \varphi$, and
- μ is satisfiable in $\mathcal{L}$.

– Search decomposed into two orthogonal components:

- Purely propositional: search for a truth assignments μ propositionally satisfying φ
- Purely domain-dependent: verify the satisfiability in $\mathcal{L}$ of μ.

Example

$$
\begin{aligned}
\varphi = \quad & \{\neg(2v_2 - v_3 > 2) \lor A_1\} \land \\
& \{\neg A_2 \lor (2v_1 - 4v_5 > 3)\} \land \\
& \{(3v_1 - 2v_2 \leq 3) \lor A_2\} \land \\
& \{\neg(2v_3 + v_4 \geq 5) \lor \neg(3v_1 - v_3 \leq 6) \lor \neg A_1\} \land \\
& \{A_1 \lor (3v_1 - 2v_2 \leq 3)\} \land \\
& \{(v_1 - v_5 \leq 1) \lor (v_5 = 5 - 3v_4) \lor \neg A_1\} \land \\
& \{A_1 \lor (v_3 = 3v_5 + 4) \lor A_2\}.
\end{aligned}
$$

$$\mu = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1), \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4)\}.$$

$$\mu' = \{\neg(2v_2 - v_3 > 2), \neg A_2, \neg A_1, (3v_1 - 2v_2 \leq 3), (v_3 = 3v_5 + 4)\}.$$

– $\mu \models_p \varphi$, but is unsatisfiable, as contains conflict sets:

$$
\begin{aligned}
& \{(3v_1 - 2v_2 \leq 3), \neg(2v_2 - v_3 > 2), \neg(3v_1 - v_3 \leq 6)\} \\
& \{(v_1 - v_5 \leq 1), (v_3 = 3v_5 + 4), \neg(3v_1 - v_3 \leq 6)\}
\end{aligned}
$$

– $\mu' \models_p \varphi$, and is satisfiable ($v_1, v_2, v_3 := 0,\ v_5 := -4/3$).

Complete collection of assignments

A collection $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ of (possibly partial) assignments propositionally satisfying φ is complete iff

$$\models_p \varphi \leftrightarrow \bigvee_j \mu_j. \qquad (1)$$

- for every total assignment η s.t. $\eta \models_p \varphi$, there is $\mu_i \in \mathcal{M}$ s.t. $\mu_i \subseteq \eta$.
  $\Longrightarrow \mathcal{M}$ represents all assignments.
- $\mathcal{M}$ “compact” representation of the whole set of total assignments propositionally satisfying φ.

Complete collection of assignments and satisfiability in $\mathcal{L}$

Proposition. Let $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ be a complete collection of truth assignments propositionally satisfying φ. Then φ is satisfiable if and only if μ_j is satisfiable for some $\mu_j \in \mathcal{M}$.

- Search decomposed into two orthogonal components:
  - Purely propositional: generate (in a lazy way) a complete collection $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ of truth assignments propositionally satisfying φ;
  - Purely domain-dependent: check one by one the satisfiability in $\mathcal{L}$ of the μ_i's.

Redundancy of complete collection of assignments

A complete collection $\mathcal{M} = \{\mu_1, \ldots, \mu_n\}$ of assignments propositionally satisfying φ is

- strongly non redundant iff, for every $\mu_i, \mu_j \in \mathcal{M}$, $(\mu_i \wedge \mu_j)$ is propositionally unsatisfiable,
- non redundant iff, for every $\mu_j \in \mathcal{M}$, $\mathcal{M} \setminus \{\mu_j\}$ is no more complete,
- redundant otherwise.

- If $\mathcal{M}$ is redundant, then $\mu_j \supseteq \mu_i$ for some $\mu_i, \mu_j \in \mathcal{M}$:

$$\begin{array}{lcll}
\models_p \varphi \leftrightarrow \bigvee_{i \neq j} \mu_i & \Longrightarrow & \models_p \bigvee_i \mu_i \leftrightarrow \bigvee_{i \neq j} \mu_i & \Longrightarrow \\
\bigvee_i \mu_i \models_p \bigvee_{i \neq j} \mu_i & \Longrightarrow & \mu_j \models_p \bigvee_{i \neq j} \mu_i & \Longrightarrow \\
\mu_j \models_p \mu_i \;\; for\ some\ i & \Longrightarrow & \mu_j \supseteq \mu_i &
\end{array}$$

- If $\mathcal{M}$ is strongly non redundant, then $\mathcal{M}$ is non redundant:

$$\begin{array}{ll}
\mu_j \wedge \mu_i \;\; propositionally\ inconsistent & \Longrightarrow \\
\mu_j \models_p \neg \mu_i & \Longrightarrow \\
\mathcal{M}\ non\ redundant &
\end{array}$$

Redundancy: example

Let $\varphi := (\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$, α, β, γ atoms. Then

1. $\{\{\alpha,\beta,\gamma\},\{\alpha,\beta,\neg\gamma\},\{\alpha,\neg\beta,\gamma\},\{\alpha,\neg\beta,\neg\gamma\},$ $\{\neg\alpha,\beta,\gamma\},\{\neg\alpha,\beta,\neg\gamma\}\}$ is the set of all total assignments propositionally satisfying φ;
2. $\{\{\alpha\},\{\alpha,\beta\},\{\alpha,\neg\gamma\},\{\alpha,\beta\},\{\beta\},\{\beta,\neg\gamma\},\{\alpha,\gamma\},\{\beta,\gamma\}\}$ is complete but redundant;
3. $\{\{\alpha\},\{\beta\}\}$ is complete, non redundant but not strongly non redundant;
4. $\{\{\alpha\},\{\neg\alpha,\beta\}\}$ is complete and strongly non redundant.

A Generalized Search Procedure

Truth assignment enumerator

A truth assignment enumerator is a total function $\textsc{Assign\_Enumerator}()$ which takes as input a formula φ in $\mathcal{L}$ and returns a complete collection $\{\mu_1, \ldots, \mu_n\}$ of assignments propositionally satisfying φ.

– A truth assignment enumerator is
  - strongly non-redundant if $\textsc{Assign\_Enumerator}(\varphi)$ is strongly non-redundant, for every φ,
  - non-redundant if $\textsc{Assign\_Enumerator}(\varphi)$ is non-redundant, for every φ,
  - redundant otherwise.

Truth assignment enumerator w.r.t. SAT solver

Remark. Notice the difference:

– A SAT solver has to find only one satisfying assignment —or to decide there is none;

– A Truth assignment enumerator has to find a complete collection of satisfying assignments.

A generalized procedure

```
boolean $\mathcal{L}$-SAT($formula\ \varphi, assignment\ \&\ \mu, model\ \&\ M$)
    do
        $\mu :=$ NEXT_ASSIGNMENT($\varphi$)                /* next in $\{\mu_1, ..., \mu_n\}$ */
        if ($\mu \neq Null$)
            $satifiable := \mathcal{L}$-SOLVE($\mu, M$);
    while ((satifiable = False) and ($\mu \neq Null$))
    if ($satifiable \neq$ False)
    then return True;                               /* a satisf. assignment found */
    else return False;                              /* no satisf. assignment found */
```

$\mathcal{L}$-SAT

- $\mathcal{L}$-SAT(φ) terminating, correct and complete $\Longleftrightarrow$ $\mathcal{L}$-SOLVE(μ) terminating, correct and complete.
- $\mathcal{L}$-SAT depends on $\mathcal{L}$ only for $\mathcal{L}$-SOLVE
- $\mathcal{L}$-SAT requires polynomial space iff
  - $\mathcal{L}$-SOLVE requires polynomial space and
  - ASSIGN_ENUMERATOR is lazy

Mandatory requirements for an assignment enumerator

An assignment enumerator must always:

- (Termination) terminate
- (Correctness) generate assignments propositionally satisfying φ
- (Completeness) generate complete set of assignments

Mandatory requirements for $\mathcal{L}$-SOLVE()

$\mathcal{L}$-SOLVE() must always:

- (Termination) terminate
- (Correctness & completeness) return $True$ if μ is satisfiable in $\mathcal{L}$, $False$ otherwise

Efficiency requirements for an assignent enumerator

To achieve the maximum efficiency, an assignent enumerator should:

- (Laziness) generate the assignments one-at-a-time.
- (Polynomial Space) require only polynomial space
- (Strong Non-redundancy) be strongly non-redundant
- (Time efficiency) be fast
- [(Symbiosis with $\mathcal{L}$-SOLVE) be able to tale benefit from failure & success information provided by $\mathcal{L}$-SOLVE (e.g., conflict sets, inferred assignments)]

Benefits of (strongly) non-redundant generators

- Non-redundant enumerators avoid generating partial assignments whose unsatisfiability is a propositional consequence of those already generated.
- Strongly non-redundant enumerators avoid generating partial assignments covering areas of the search space which are covered by already-generated ones.
- Strong non-redundancy provides a logical warrant that an already generated assignment will never be generated again. $\Longrightarrow$no extra control required to avoid redundancy.

Efficiency requirements for $\mathcal{L}$-SOLVE()

To achieve the maximum efficiency, $\mathcal{L}$-SOLVE() should:

- (Time efficiency) be fast
- (Polynomial Space) require only polynomial space
- [(Symbiosis with ASSIGN_ENUMERATOR) be able to produce failure & success information (e.g., conflict sets, inferred assignments)]
- [(Incrementality) be incremental: $\mathcal{L}$-SOLVE$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{L}$-SOLVE(μ_1)]

Extending existing SAT procedures

General ideas

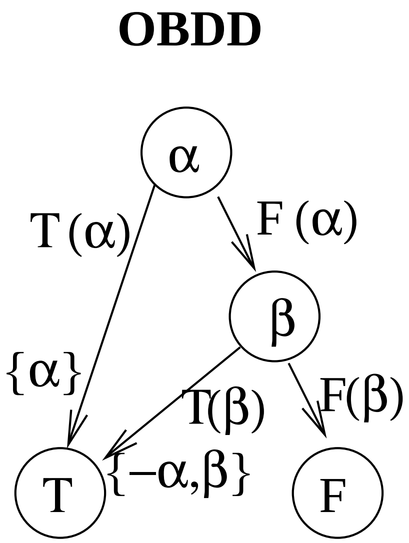
Existing SAT procedures are natural candidates to be used as assignment enumerators.

- Atoms labelled by propositional atoms
- Slight modifications
  (backtrack when assignment found)
- Completeness to be verified!
  (E.g., DPLL with Pure literal)
- Candidates: OBDDs, Semantic Tableaux, DPLL

OBDDs

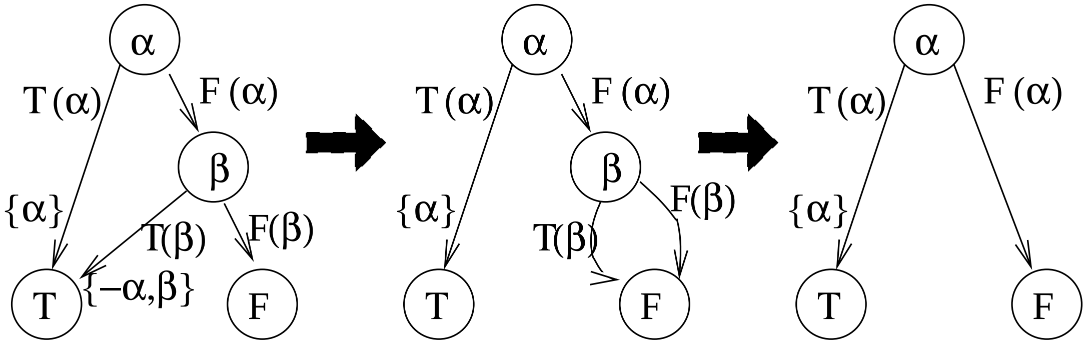
- In an OBDD, the set of paths from the root to (1) represent a complete collection of assignments
- Some may be inconsistent in $\mathcal{L}$
- Reduction: [21, 60, 2]
  1. inconsistent paths from the root to internal nodes are detected
  2. they are redirected to the (0) node
  3. the resulting OBDD is simplified.

OBDD: example



OBDD of $(\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$.

OBDD reduction: example



Reduced OBDD of $(\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$, $\alpha := (x - y \leq 4)$, $\beta := (x - y \leq 2)$.

OBDD: summary

- strongly non-redundant
- time-efficient
- factor sub-graphs
- require exponential memory
- non lazy
- [allow for early pruning]
- [do not allow for backjumping or learning]

Generalized semantic tableaux

- General rules = propositional rules + $\mathcal{L}$-specific rules

$$\left\{ \begin{array}{ccc} \dfrac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}} & \dfrac{\neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\neg\varphi_1\\ \neg\varphi_2\end{array}} & \dfrac{\neg(\varphi_1 \to \varphi_2)}{\begin{array}{c}\varphi_1\\ \neg\varphi_2\end{array}} \\ & \dfrac{\neg\neg\varphi}{\varphi} & \\ \dfrac{\varphi_1 \vee \varphi_2}{\varphi_1 \quad \varphi_2} & \dfrac{\neg(\varphi_1\wedge\varphi_2)}{\neg\varphi_1 \quad \neg\varphi_2} & \dfrac{\varphi_1\to\varphi_2}{\neg\varphi_1 \quad \varphi_2} \\ \dfrac{\varphi_1 \leftrightarrow \varphi_2}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \varphi_2 & \neg\varphi_2\end{array}} & \dfrac{\neg(\varphi_1\leftrightarrow\varphi_2)}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \neg\varphi_2 & \varphi_2\end{array}} & \end{array} \right\} \cup \left\{ \begin{array}{c} \mathcal{L}\text{-specific} \\ \text{Rules}\end{array} \right\}$$

- Widely used by logicians

Generalized tableau algorithm

```
function $\mathcal{L}$-Tableau($\Gamma$)
    if $A_i \in \Gamma$ and $\neg A_i \in \Gamma$                          /* branch closed */
        then return False;
    if $(\varphi_1 \wedge \varphi_2) \in \Gamma$                             /* ∧-elimination */
        then return $\mathcal{L}$-Tableau($\Gamma \cup \{\varphi_1, \varphi_2\} \setminus \{(\varphi_1 \wedge \varphi_2)\}$);
    if $(\neg\neg\varphi_1) \in \Gamma$                                  /* ¬¬-elimination */
        then return $\mathcal{L}$-Tableau($\Gamma \cup \{\varphi_1\} \setminus \{(\neg\neg\varphi_1)\}$);
    if $(\varphi_1 \vee \varphi_2) \in \Gamma$                             /* ∨-elimination */
        then return     $\mathcal{L}$-Tableau($\Gamma \cup \{\varphi_1\} \setminus \{(\varphi_1 \vee \varphi_2)\}$)  or
                        $\mathcal{L}$-Tableau($\Gamma \cup \{\varphi_2\} \setminus \{(\varphi_1 \vee \varphi_2)\}$);
    ...
    return ($\mathcal{L}$-SOLVE($\Gamma$)= satisfiable);                /* branch expanded */
```

General tableaux: example

Tableau Search Graph

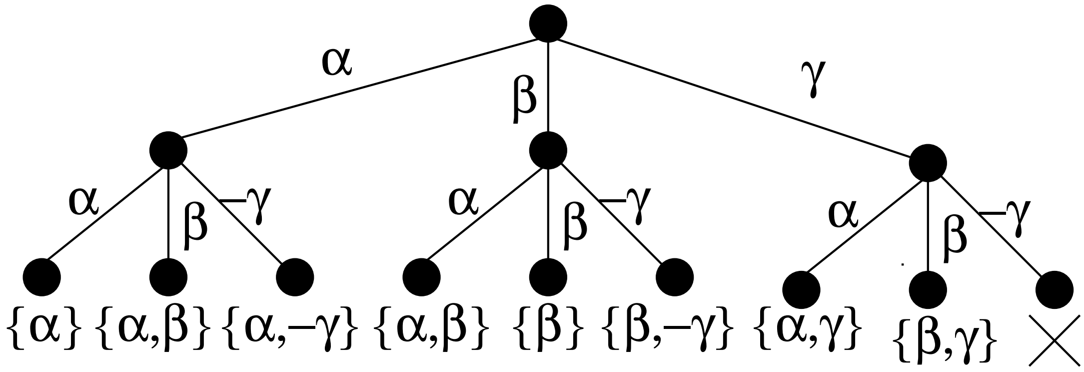


Tableau search graph for $(\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$.

Generalized tableaux: problems

Two main problems [25, 42, 43]

- syntactic branching
  - branch on disjunctions
  - possible many duplicate or subsumed branches $\Longrightarrow$redundant
  - duplicates search (both propositional and domain-dependent)
- no constraint violation detection
  - incapable to detect when current branches violate a constraint $\Longrightarrow$lots of redundant propositional search.

Syntactic branching: example

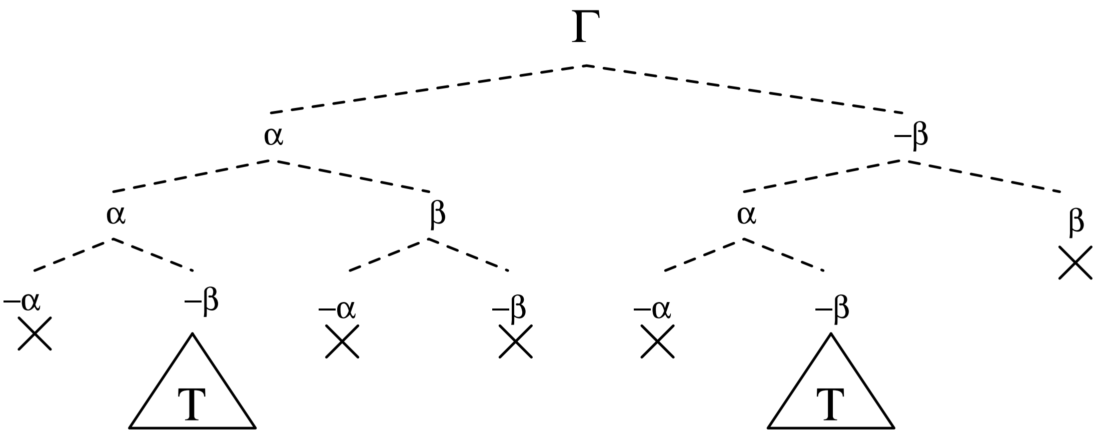


Tableau search graph for $(\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$.

Detecting constraints violations: example

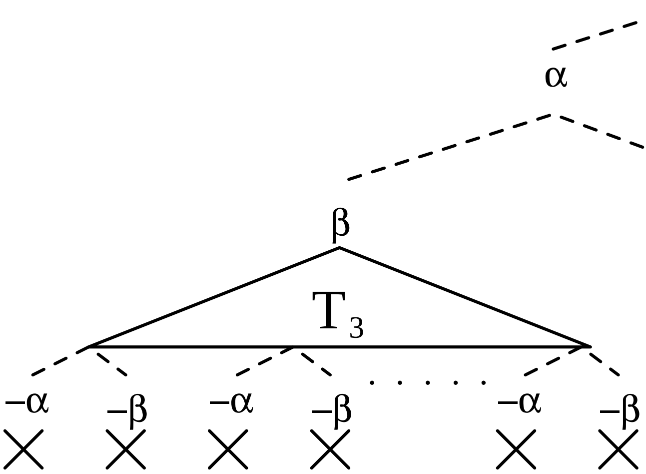


Tableau search graph for $(\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta)$

Generalized tableaux: summary

– lazy

– require polynomial memory

– redundant

– time-inefficient

– [allow backjumping]

– [do not allow learning]

Tableaux: remark

The word "Tableau" is a bit overloaded in literature. Some existing (and rather efficient) systems, like FacT, DLP [48] and RACER [91], call themselves "Tableau" procedures, although they use a DPLL-like technique to perform boolean reasoning.

"(...) DLP deals with non-determinism in the model construction algorithm by performing a semantic branching search, as in the Davis-Putnam-Logemann-Loveland procedure (DPLL), instead of the syntactic branching search used by most earlier tableaux based implementations (...)" [68]

"(...) The RACER architecture incorporates the following standard optimization techniques: dependency-directed backtracking (...) and DPLL-style semantic branching (...)" [91]

Same for the boolean system KE [25] and its derived systems.

Generalized DPLL

- General rules = propositional rules + $\mathcal{L}$-specific rules

$$\left\{ \begin{array}{ll} \dfrac{\varphi_1 \wedge (l) \wedge \varphi_2}{(\varphi_1 \wedge \varphi_2)[l|\top]} & (Unit) \\ \\ \dfrac{\varphi}{\varphi[l|\top] \quad \varphi[l|\bot]} & (split) \end{array} \right\} \cup \left\{ \begin{array}{c} \mathcal{L}\text{-specific} \\ \text{Rules} \end{array} \right\}$$

- Equivalent formalism described in [90]
- NOTE: No Pure Literal Rule (on non-boolean atoms):
  Pure literal causes incomplete assignment sets!
- if l pure in φ, typically $\varphi[l|\top]$ is investigated before $\varphi[l|\bot]$

Pure literal and Generalized DPLL: Example

$$\begin{aligned}\varphi = \; & ((x - y \le 1) \vee A_1) \wedge \\ & ((y - z \le 2) \vee A_2) \wedge \\ & (\neg(x - z \le 4) \vee A_2) \wedge \\ & (\neg A_2 \vee A_3) \wedge \\ & (\neg A_2 \vee \neg A_3)\end{aligned}$$

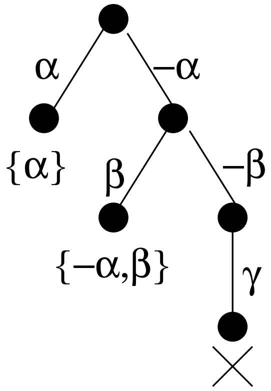
- A satisfiable assignment propositionally satisfying φ is: $\mu = \{A_1, \neg A_2, (y - z \le 2), \neg(x - z \le 4)\}$
- No satisfiable assignment propositionally satisfying φ contains $(x - y \le 1)$
- Pure literal may assign $(x - y \le 1) := \top$ as first step $\Longrightarrow$return unsatisfiable.

Generalized DPLL algorithm

```
function $\mathcal{L}$-DPLL($\varphi, \mu$)
    if $\varphi = \top$                                          /* base      */
        then return ($\mathcal{L}$-SOLVE($\mu$)=satisfiable);
    if $\varphi = \bot$                                          /* backtrack */
        then return False;
    if {a unit clause $(l)$ occurs in $\varphi$}                 /* unit      */
        then return $\mathcal{L}$-DPLL(assign$(l, \varphi), \mu \wedge l$);
    l := choose-literal($\varphi$);                              /* split     */
    return  $\mathcal{L}$-DPLL(assign$(l, \varphi), \mu \wedge l$)  or
            $\mathcal{L}$-DPLL(assign$(\neg l, \varphi), \mu \wedge \neg l$);
```

General DPLL: example

DPLL search graph



DPLL search graph for $(\alpha \vee \beta \vee \gamma) \wedge (\alpha \vee \beta \vee \neg\gamma)$.

Generalized DPLL vs. generalized tableaux

Two big advantages: [25, 42, 43]

- semantic vs. syntactic branching
  - branch on truth values
  - no duplicate or subsumed branches $\Longrightarrow$strongly non redundant
  - no search duplicates
- constraint violation detection
  - backtracks as soon as the current branch violates a constraint $\Longrightarrow$no redundant propositional search.

Semantic branching: example

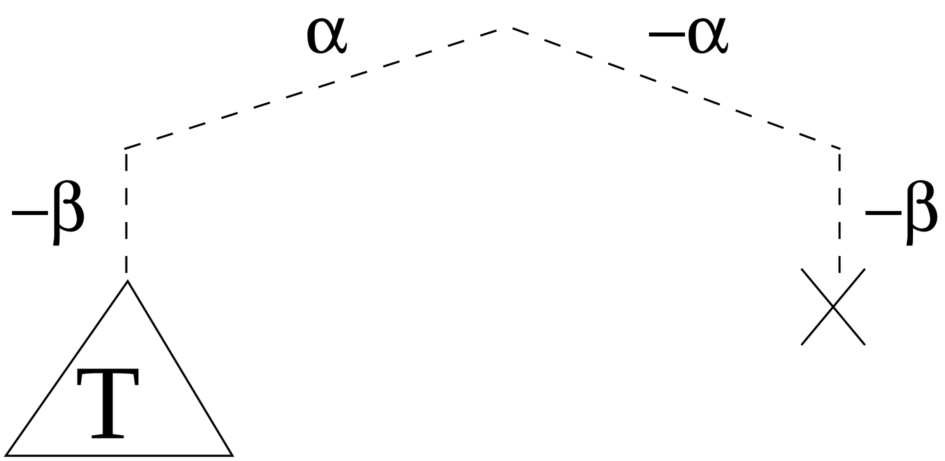
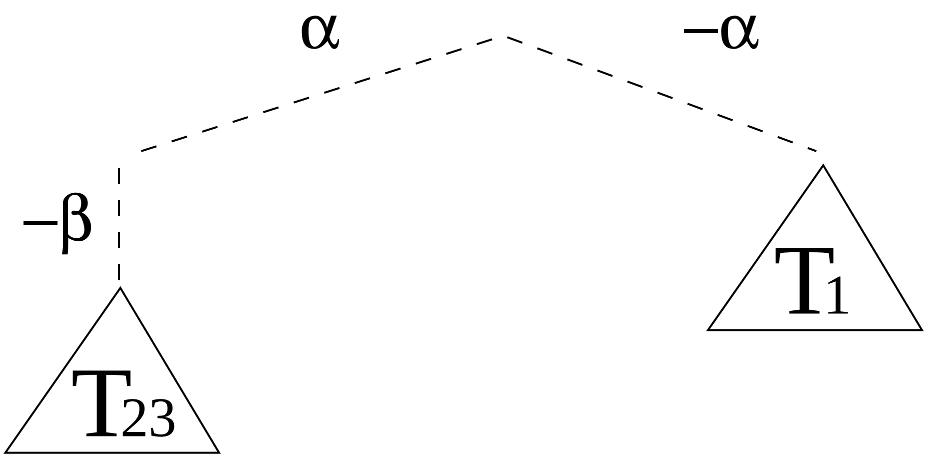


Tableau search graph for $(\alpha \vee \neg\beta) \wedge (\alpha \vee \beta) \wedge (\neg\alpha \vee \neg\beta)$.

Detecting constraints violations: example



DPLL search graph for $(\alpha \vee \phi_1) \wedge (\beta \vee \phi_2) \wedge \phi_3 \wedge (\neg\alpha \vee \neg\beta)$

Generalized DPLL vs. generalized tableaux: remarks

- ▷ Generalized tableaux reason on subformula instances
- ▷ Generalized DPLL reasons on atoms
  $\Longrightarrow$all instances of an atom are handled contemporarily
- ▷ If the atoms have no multiple occurrences, the benefits of DPLL vs. tableaux are negligible (unless learning is used)

Generalized DPLL: summary

- lazy
- require polynomial memory
- strongly non redundant
- time-efficient
- [allow backjumping and learning]

Optimizations

Possible Improvements

- Preprocessing atoms [41, 48, 7]
- Static learning [3]
- Early pruning [41, 21, 6]
- Enhanced Early pruning [6]
- Backjumping [48, 95]
- Memoizing [48, 37]
- Learning [48, 95]
- Forward Checking [3]
- Triggering [95, 6]
- ...

Preprocessing atoms [41, 48, 7]

Source of inefficiency: semantically equivalent but syntactically different atoms are not recognized to be identical [resp. one the negation of the other] $\Longrightarrow$they may be assigned different [resp. identical] truth values.

Solution: rewrite trivially equivalent atoms into one.

Preprocessing atoms (cont.)

- Sorting: $(v_1 + v_2 \leq v_3 + 1)$, $(v_2 + v_1 \leq v_3 + 1)$, $(v_1 + v_2 - 1 \leq v_3) \Longrightarrow (v_1 + v_2 - v_3 \leq 1))$;
- Rewriting dual operators: $(v_1 < v_2)$, $(v_1 \geq v_2) \Longrightarrow (v_1 < v_2)$, $\neg(v_1 < v_2)$
- Exploiting associativity: $(v_1 + (v_2 + v_3) = 1)$, $((v_1 + v_2) + v_3) = 1) \Longrightarrow (v_1 + v_2 + v_3 = 1)$;
- Factoring $(v_1 + 2.0v_2 \leq 4.0)$, $(-2.0v_1 - 4.0v_2 \geq -8.0)$, $\Longrightarrow (0.25v_1 + 0.5v_2 \leq 1.0)$;
- Exploiting properties of $\mathcal{L}$: $(v_1 \leq 3)$, $(v_1 < 4) \Longrightarrow (v_1 \leq 3)$ if $v_1 \in \mathbb{Z}$;
- ...

Preprocessing atoms: summary

- Very efficient with DPLL
- Presumably very efficient with OBDDs
- Scarcely efficient with semantic tableaux

Static learning [3]

- Rationale: Many literals are mutually exclusive (e.g., $(x - y < 3), \neg(x - y < 5)$)
- Preprocessing step: detect these literals and add binary clauses to the input formula: (e.g., $\neg(x - y < 3) \vee (x - y < 5)$)
- (with DPLL) assignments including both literals are never generated.
- requires $O(|\varphi|^2)$ steps.

Static learning (cont.)

- Very efficient with DPLL
- Possibly very efficient with OBDDs (?)
- Completely ineffective with semantic tableaux

Early pruning [41, 21, 6]

- rationale: if an assignment μ' is unsatisfiable, then all its extensions are unsatisfiable.
- the unsatisfiability of μ' detected during its construction, $\Longrightarrow$avoids checking the satisfiability of all the up to $2^{|Atoms(\varphi)|-|\mu'|}$ assignments extending μ'.
- Introduce a satisfiability test on incomplete assignments just before every branching step:

```
if Likely-Unsatisfiable($\mu$)            /* early pruning */
    if ($\mathcal{L}$-SOLVE($\mu$) = $False$)
        then return False;
```

DPLL+Early pruning

```
function $\mathcal{L}$-DPLL($\varphi, \mu$)
    if $\varphi = \top$                                              /* base          */
        then return ($\mathcal{L}$-SOLVE($\mu$)=satisfiable);
    if $\varphi = \bot$                                              /* backtrack */
        then return False;
    if {a unit clause $(l)$ occurs in $\varphi$}                     /* unit          */
        then return $\mathcal{L}$-DPLL($assign(l, \varphi), \mu \wedge l$);
    if Likely-Unsatisfiable($\mu$)                                   /* early pruning */
        if ($\mathcal{L}$-SOLVE($\mu$) = False)
            then return False;
    l := choose-literal($\varphi$);                                  /* split         */
    return  $\mathcal{L}$-DPLL($assign(l, \varphi), \mu \wedge l$)  or
            $\mathcal{L}$-DPLL($assign(\neg l, \varphi), \mu \wedge \neg l$);
```

Early pruning: example

$$
\begin{aligned}
\varphi = \quad & \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\
& \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\
& \{(3v_1 - 2v_2 \leq 3) \vee A_2\} \wedge \\
& \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (3v_1 - 2v_2 \leq 3)\} \wedge \\
& \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\
& \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\}.
\end{aligned}
$$

– Suppose it is built the intermediate assignment:

$$\mu' = \neg(2v_2 - v_3 > 2) \wedge \neg A_2 \wedge (3v_1 - 2v_2 \leq 3) \wedge \neg(3v_1 - v_3 \leq 6).$$

– If $\mathcal{L}$-SOLVE is invoked on μ', it returns *False*, and $\mathcal{L}$-DPLL backtracks without exploring any extension of μ'.

Early pruning: drawback

- Reduces drastically the search
- Drawback: possibly lots of useless calls to $\mathcal{L}$-SOLVE
  $\Longrightarrow$to be used with care when $\mathcal{L}$-SOLVE calls recursively $\mathcal{L}$-SAT (e.g., with modal logics)
- Roughly speaking, worth doing when each branch saves at least one branching
- Possible solutions:
  - introduce a selective heuristic Likely-unsatisfiable
  - use incremental versions of $\mathcal{L}$-SOLVE

  one split.

Early pruning: Likely-unsatisfiable

- Rationale: if no literal which may likely cause conflict with the previous assignment has been added since last call, return false.
- Examples: return false if they are added only
  - boolean literals
  - disequalities $(x - y \neq 3)$
  - atoms introducing new variables $(x - z \neq 3)$
  - ...

Early pruning: incrementality of $\mathcal{L}$-SOLVE

- With early pruning, lots of incremental calls to $\mathcal{L}$-SOLVE:

$$\begin{array}{lll} \mathcal{L}\text{-SOLVE}(\mu) & \Longrightarrow & \text{satisfiable} \\ \mathcal{L}\text{-SOLVE}(\mu \cup \mu') & \Longrightarrow & \text{satisfiable} \\ \mathcal{L}\text{-SOLVE}(\mu \cup \mu' \cup \mu'') & \Longrightarrow & \text{satisfiable} \\ \ldots & & \end{array}$$

- $\mathcal{L}$-SOLVE incremental: $\mathcal{L}$-SOLVE$(\mu_1 \cup \mu_2)$ reuses computation of $\mathcal{L}$-SOLVE(μ_1) without restarting from scratch $\Longrightarrow$ lots of computation saved
- requires saving the status of $\mathcal{L}$-SOLVE

Early pruning: summary

- Very efficient with DPLL & OBDDs
- Possibly very efficient with semantic tableaux (?)
- In some cases may introduce big overhead (e.g., modal logics)
- Benefits if $\mathcal{L}$-SOLVE is incremental

Enhanced Early Pruning [6]

- In early pruning, $\mathcal{L}$-SOLVE is not effective if it returns "satisfiable".
- $\mathcal{L}$-SOLVE(μ) may be able to derive (easily) a sub-assignment η s.t. $\mu \models \eta$, and return it.
- The literals in η are then unit-propagated away.

Enhanced Early Pruning: Examples

(We assume that all the following literals occur in φ.)

- If $(v_1 - v_2 \leq 4) \in \mu$ and $(v_1 - v_2 \leq 6) \notin \mu$, then $\mathcal{L}$-SOLVE can derive $(v_1 - v_2 \leq 6)$ from μ.
- If $(v_1 - v_3 > 2), (v_2 = v_3) \in \mu$ and $(v_1 - v_2 > 2) \notin \mu$, then $\mathcal{L}$-SOLVEcan derive $(v_1 - v_2 > 2)$ from μ.

Enhanced Early Pruning: summary

- Further improves efficiency with DPLL
- Presumably scarcely effective with semantic tableaux
- Effective with OBDDs?
- Requires a sophisticated $\mathcal{L}$-SOLVE (able to perform deductive inference)

Backjumping (driven by $\mathcal{L}$-SOLVE) [48, 95]

- Similar to SAT backjumping
- Rationale: same as for early pruning
- Idea: when a branch is found unsatisfiable in $\mathcal{L}$,
  1. $\mathcal{L}$-SOLVE returns the conflict set causing the failure
  2. $\mathcal{L}$-SAT backtracks to the most recent branching point in the conflict set

Backjumping: Example

$$\begin{aligned}\varphi = \quad & \{\neg(2v_2 - v_3 > 2) \vee A_1\} \wedge \\ & \{\neg A_2 \vee (2v_1 - 4v_5 > 3)\} \wedge \\ & \{(3v_1 - 2v_2 \leq 3) \vee A_2\} \wedge \\ & \{\neg(2v_3 + v_4 \geq 5) \vee \neg(3v_1 - v_3 \leq 6) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (3v_1 - 2v_2 \leq 3)\} \wedge \\ & \{(v_1 - v_5 \leq 1) \vee (v_5 = 5 - 3v_4) \vee \neg A_1\} \wedge \\ & \{A_1 \vee (v_3 = 3v_5 + 4) \vee A_2\}.\end{aligned}$$

$$\mu = \{\neg(2v_2 - v_3 > 2), \neg A_2, (3v_1 - 2v_2 \leq 3), (v_1 - v_5 \leq 1), \neg(3v_1 - v_3 \leq 6), (v_3 = 3v_5 + 4)\}.$$

- $\mathcal{L}\text{-SOLVE}(\mu)$ returns *false* with the conflict set:

$$\{(3v_1 - 2v_2 \leq 3), \neg(2v_2 - v_3 > 2), \neg(3v_1 - v_3 \leq 6)\}$$

- $\mathcal{L}\text{-SAT}$ can jump back directly to the branching point $\neg(3v_1 - v_3 \leq 6)$, without branching on $(v_3 = 3v_5 + 4)$.

Backjumping vs. Early Pruning

- Backjumping requires no extra calls to $\mathcal{L}\text{-SOLVE}$
- Effectiveness depends on the conflict set C, i.e., on how recent the most recent branching point in C is.
- Example: no pruning effect with the conflict set:

$$\{(v_1 - v_5 \leq 1), (v_3 = 3v_5 + 4), \neg(3v_1 - v_3 \leq 6)\}$$

- Same pruning effect as with Early Pruning only with the best conflict set
- More effective than Early Pruning only when the overhead compensates the pruning effect (e.g., modal logics with high depths).

Backjumping: summary

- Very efficient with DPLL
- Never applied to OBDDs
- Very efficient with semantic tableaux
- Alternative to but less effective than early pruning.
- No significant overhead
- $\mathcal{L}$-SOLVE must be able to detect conflict sets.

Memoizing [48, 37]

- Idea 1:
  - When a conflict set C is revealed, then C can be cached into an ad hoc data structure
  - $\mathcal{L}$-SOLVE(μ) checks first if (any subset of) μ is cached. If yes, returns unsatisfiable.
- Idea 2:
  - When a satisfying (sub)-assignment μ' is found, then μ' can be cached into an ad hoc data structure
  - $\mathcal{L}$-SOLVE(μ) checks first if (any superset of) μ is cached. If yes, returns satisfiable.

Memoizing (cont.)

- Can dramatically prune search.
- May cause a blowup in memory.
- Applicable also to semantic tableaux.
- Idea 1 subsumed by learning.

Learning (driven by $\mathcal{L}$-SOLVE) [48, 95]

- Similar to SAT learning
- Idea: When a conflict set C is revealed, then $\neg C$ can be added to the clause set
  $\Longrightarrow$DPLL will never again generate an assignment containing C.
- May avoid a lot of redundant search.
- Problem: may cause a blowup in space
  $\Longrightarrow$techniques to control learning and to drop learned clauses when necessary

Learning: example

- $\mathcal{L}$-SOLVE returns the conflict set:
  $\{(3v_1 - 2v_2 \le 3), \neg(2v_2 - v_3 > 2), \neg(3v_1 - v_3 \le 6)\}$
- it is added the clause
  $\neg(3v_1 - 2v_2 \le 3) \vee (2v_2 - v_3 > 2) \vee (3v_1 - v_3 \le 6)$
- Prunes up to 2^{N-3} assignments
  $\Longrightarrow$the smaller the conflict set, the better.

Learning: summary

- Very efficient with DPLL
- Never applied to OBDDs
- Completely ineffective with semantic tableaux
- May cause memory blowup
- $\mathcal{L}$-SOLVE must be able to detect conflict sets.

Forward Checking [3]

- Idea: if $\mu \wedge l \wedge l'$ inconsistent, then $\mu \wedge l \models \neg l'$
- $assign(\varphi, l)$ substituted with $fc\_assign(\varphi, \mu \wedge l)$: $fc\_assign(\varphi, \mu \wedge l)$ replaces $cl \vee l'$ with cl if $\mathcal{L}\text{-SOLVE}(\mu \wedge l \wedge l')$ returns false, for every l'
- can significantly prune search
- significant overhead: many possibly redundant calls to $\mathcal{L}$-SOLVE

Triggering [95, 6]

Proposition Let C be a non-boolean atom occurring only positively [resp. negatively] in φ. Let $\mathcal{M}$ be a complete set of assignments for φ, and let

$$\mathcal{M}' := \{\mu_j/\neg C\} | \mu_j \in \mathcal{M}\} \quad [resp.\ \{\mu_j/C\} | \mu_j \in \mathcal{M}\}].$$

Then φ is satisfiable if and only if there exist a satisfiable $\eta' \in \mathcal{M}'$ s.t. $\eta' \models_p \varphi$.

Proof (Sketch) The "if" case is trivial. If φ is satisfiable, then there is a satisfiable $\eta \in \mathcal{M}$ s.t. $\eta \models_p \varphi$ because $\mathcal{M}$ is complete. If $\neg C \notin \eta$, then the thesis holds with $\eta' := \eta$. If $\neg C \in \eta$, then let $\eta' := \eta/\neg C$. η' is trivially satisfiable. As $\eta \models_p \varphi$ and C occurs only positively in φ, $\eta' \models_p \varphi$.

Triggering (cont.)

- If we have non-boolean atoms occurring only positively [negatively] in φ, we can drop any negative [positive] occurrence of them from the assignment to be checked by $\mathcal{L}$-SOLVE
- Particularly useful when we deal with equality atoms (e.g., $(v_1 - v_2 = 3.2)$), as handling negative equalities like $(v_1 - v_2 \neq 3.2)$ forces splitting: $(v_1 - v_2 > 3.2) \vee (v_1 - v_2 < 3.2)$.

Application Fields

- Modal Logics [41, 48, 43, 37]
- Description Logics [42, 48]
- Boolean+Mathematical reasoning (Temporal reasoning [3], Resource Planning [95], Verification of Timed Systems [60, 6, 9, 84, 29, 65, 70], Verification of systems with arithmetical operators [21, 89], verification of hybrid systems [8]
- decision procedures in combined theories [62, 63, 81, 31, 7, 6, 12, 13, 89, 29, 56, 78, 90, 87, 86]
- ...

Case study: Modal Logic(s)

Satisfiability in Modal logics

- Propositional logics enhanced with modal operators $\Box_i$, K_i, etc.
- Used to represent complex concepts like knowledge, necessity/possibility, etc.
- Based on Kripke's possible worlds semantics [54]
- Very hard to decide [45, 44] (typically PSPACE-complete or worse)
- Strictly related to Description Logics [72] (ex: $K(m) \Longleftrightarrow \mathcal{ALC}$)
- Various fields of application: AI, formal verification, knowledge bases, etc.

Syntax

Given a non-empty set of primitive propositions $\mathcal{A} = \{A_1, A_2, \ldots\}$ and a set of m modal operators $\mathcal{B} = \{\Box_1, \ldots, \Box_m\}$, the modal language $\mathcal{L}$ is the least set of formulas containing $\mathcal{A}$, closed under the set of propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$ and the set of modal operators in $\mathcal{B}$.

- depth(φ) is the maximum number of nested modal operators in φ.
- "$\Box_i\varphi$" can be interpreted as "Agent i knows φ"

Semantics

- A Kripke structure for $\mathcal{L}$ is a tuple $M = < \mathcal{U}, \pi, \mathcal{R}_1, \ldots, \mathcal{R}_m >$, where
  - $\mathcal{U}$ is a set of states $u_1, u_2, \ldots$
  - π is a function $\pi : \mathcal{A} \times \mathcal{U} \longmapsto \{\top, \bot\}$,
  - each $\mathcal{R}_r$ is a binary relation on the states of $\mathcal{U}$.

Semantics (cont)

Given M, u s.t. $u \in \mathcal{U}$, $M, u \models \varphi$ is defined as follows:

$$
\begin{array}{lcl}
M, u \models A_i,\ A_i \in \mathcal{A} & \Longleftrightarrow & \pi(A_i, u) = \top; \\
M, u \models \neg\varphi_1 & \Longleftrightarrow & M, u \not\models \varphi_1; \\
M, u \models \varphi_1 \wedge \varphi_2 & \Longleftrightarrow & M, u \models \varphi_1 \textit{ and } M, u \models \varphi_2; \\
M, u \models \varphi_1 \vee \varphi_2 & \Longleftrightarrow & M, u \models \varphi_1 \textit{ or } M, u \models \varphi_2. \\
\ldots & & \\
M, u \models \Box_r\varphi_1,\ \Box_r \in \mathcal{B} & \Longleftrightarrow & M, v \models \varphi_1 \text{ for every } v \in \mathcal{U} \\
 & & \text{s.t. } \mathcal{R}_r(u, v) \text{ holds in } M. \\
M, u \models \neg\Box_r\varphi_1,\ \Box_r \in \mathcal{B} & \Longleftrightarrow & M, v \models \neg\varphi_1 \text{ for some } v \in \mathcal{U} \\
 & & \text{s.t. } \mathcal{R}_r(u, v) \text{ holds in } M.
\end{array}
$$

Semantics (cont)

The (normal) modal logics vary with the properties of $\mathcal{R}_r$:

Axiom	Property of $\mathcal{R}$	Description
B	symmetric	$\forall\, u\, v\ \mathcal{R}(u,v) \Longrightarrow \mathcal{R}(v,u)$
D	serial	$\forall\, u\, \exists\, v\ \mathcal{R}(u,v)$
T	reflexive	$\forall\, u\ \mathcal{R}(u,u)$
4	transitive	$\forall\, u\, v\, w\ \mathcal{R}(u,v)\ e\ \mathcal{R}(v,w) \Longrightarrow \mathcal{R}(u,w)$
5	euclidean	$\forall\, u\, v\, w\ \mathcal{R}(u,v)\ e\ \mathcal{R}(u,w) \Longrightarrow \mathcal{R}(v,w)$

Normal Modal Logic	Properties of $\mathcal{R}_r$
K	—
KB	symmetric
KD	serial
KT = KDT (T)	reflexive
K4	transitive
K5	euclidean
KBD	symmetric and serial
KBT = KBDT (B)	symmetric and reflexive
KB4 = KB5 = KB45	symmetric and transitive
KD4	serial and transitive
KD5	serial and euclidean
KT4 = KDT4 (S4)	reflexive and transitive
KT5 = KBD4 = KBD5 = KBT4 = KBT5 = KDT5 = KT45 = KBD45 = KBT45 = KDT45 = KBDT4 = KBDT5 = KBDT45 (S5)	reflexive, transitive and symmetric (equivalence)
K45	transitive and euclidean
KD45	serial, transitive and euclidean

Axiomatic framework

- Basic Axioms:

$I.\quad \alpha \to (\beta \to \alpha),$

$II.\quad (\alpha \to (\beta \to \gamma)) \to ((\alpha \to \beta) \to (\alpha \to \gamma)),$

$III.\quad (\neg\alpha \to \beta) \to ((\neg\alpha \to \neg\beta) \to \alpha),$

$K:\quad \Box_r\alpha \to (\Box_r(\alpha \to \beta) \to \Box_r\beta)$

- Specific Axioms:

$B.\quad \alpha \to \Box_r\neg\Box_r\neg\alpha,$

$D.\quad \Box_r\alpha \to \neg\Box_r\neg\alpha,$

$T.\quad \Box_r\alpha \to \alpha,$

$4.\quad \Box_r\alpha \to \Box_r\Box_r\alpha,$

$5.\quad \neg\Box_r\alpha \to \Box_r\neg\Box_r\alpha.$

Axiomatic framework (cont.)

- Inference rules:

$$\frac{\alpha \quad \alpha \to \beta}{\beta}\ \text{(modus ponens)},$$

$$\frac{\alpha}{\Box_r \alpha}\ \text{(necessitation)}.$$

- Correctness & completeness:

  φ is valid $\Longleftrightarrow$ φ can be deduced

Tableaux for modal K(m)/$\mathcal{ALC}$ [32]

- Rules = tableau rules + $K(m)$-specific rules

$$\left\{ \begin{array}{ccc}
\dfrac{\varphi_1 \wedge \varphi_2}{\begin{array}{c}\varphi_1\\ \varphi_2\end{array}} & \dfrac{\neg(\varphi_1 \vee \varphi_2)}{\begin{array}{c}\neg\varphi_1\\ \neg\varphi_2\end{array}} & \dfrac{\neg(\varphi_1 \to \varphi_2)}{\begin{array}{c}\varphi_1\\ \neg\varphi_2\end{array}} \\
 & \dfrac{\neg\neg\varphi}{\varphi} & \\
\dfrac{\varphi_1 \vee \varphi_2}{\varphi_1 \quad \varphi_2} & \dfrac{\neg(\varphi_1 \wedge \varphi_2)}{\neg\varphi_1 \quad \neg\varphi_2} & \dfrac{\varphi_1 \to \varphi_2}{\neg\varphi_1 \quad \varphi_2} \\
\dfrac{\varphi_1 \leftrightarrow \varphi_2}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \varphi_2 & \neg\varphi_2\end{array}} & \dfrac{\neg(\varphi_1 \leftrightarrow \varphi_2)}{\begin{array}{cc}\varphi_1 & \neg\varphi_1\\ \neg\varphi_2 & \varphi_2\end{array}} &
\end{array} \right\} \cup \left\{ \dfrac{\Box_r\alpha_1, \ldots, \Box_r\alpha_N, \neg\Box_r\beta_j}{\alpha_1, \ldots, \alpha_N, \neg\beta_j} \right\}$$

DPLL for K(m)/$\mathcal{ALC}$: K-SAT [41, 42]

– Rules = DPLL rules + $K(m)$-specific rules

$$\left\{ \begin{array}{l} \dfrac{\varphi_1 \wedge (l) \wedge \varphi_2}{(\varphi_1 \wedge \varphi_2)[l|\top]} \ (Unit) \\ \\ \dfrac{\varphi}{\varphi[l|\top] \quad \varphi[l|\bot]} \ (split) \end{array} \right\} \cup \left\{ \dfrac{\Box_r\alpha_1,\ldots,\Box_r\alpha_N,\neg\Box_r\beta_j}{\alpha_1,\ldots,\alpha_N,\neg\beta_j} \right\}$$

The K-SAT algorithm [41, 42]

```
function K-SAT($\varphi$)
    return K-DPLL($\varphi, \top$);

function K-DPLL($\varphi, \mu$)
    if $\varphi = \top$                                  /* base        */
        then return K-SOLVE($\mu$);
    if $\varphi = \bot$                                  /* backtrack */
        then return False;
    if {a unit clause $(l)$ occurs in $\varphi$}          /* unit        */
        then return K-DPLL($assign(l, \varphi), \mu \wedge l$);
    if Likely-Unsatisfiable($\mu$)                       /* early pruning */
        if not K-SOLVE($\mu$)
            then return False;
    $l$ := choose-literal($\varphi$);                    /* split       */
    return  K-DPLL($assign(l, \varphi), \mu \wedge l$) or
            K-DPLL($assign(\neg l, \varphi), \mu \wedge \neg l$);
```

The K-SAT algorithm (cont.)

```
function K-SOLVE($\bigwedge_i \Box_1 \alpha_{1i} \wedge \bigwedge_j \neg\Box_1 \beta_{1j} \wedge \ldots \wedge \bigwedge_i \Box_m \alpha_{mi} \wedge \bigwedge_j \neg\Box_m \beta_{mj} \wedge \gamma$)
    for each box index $r$ do
        if not K-SOLVE$_{restr}$($\bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$)
            then return False;
    return True;

function K-SOLVE$_{restr}$($\bigwedge_i \Box_r \alpha_{ri} \wedge \bigwedge_j \neg\Box_r \beta_{rj}$)
    for each conjunct "$\neg\Box_r \beta_{rj}$" do
        if not K-SAT($\bigwedge_i \alpha_{ri} \wedge \neg\beta_{rj}$)
            then return False;
    return True;
```

K-SAT: Example

$$
\begin{array}{rl}
\varphi = & \{\neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \vee A_1 \vee A_5\} \wedge \\
& \{\neg A_2 \vee \neg A_5 \vee \Box_2(\neg A_2 \vee \neg A_4 \vee \neg A_3)\} \wedge \\
& \{A_1 \vee \Box_2(\neg A_4 \vee A_5 \vee A_2) \vee A_2\} \wedge \\
& \{\neg\Box_2(A_4 \vee \neg A_3 \vee A_1) \vee \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \vee \neg A_5\} \wedge \\
& \{\neg A_3 \vee A_1 \vee \Box_2(\neg A_4 \vee A_5 \vee A_2)\} \wedge \\
& \{\Box_1(\neg A_5 \vee A_4 \vee A_3) \vee \Box_1(\neg A_1 \vee A_4 \vee A_3) \vee \neg A_1\} \wedge \\
& \{A_1 \vee \Box_1(\neg A_2 \vee A_1 \vee A_4) \vee A_2\}
\end{array}
$$

$\Downarrow$ K-SOLVE()

$$
\begin{array}{rllll}
\mu = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge & & [\bigwedge_i \Box_1 \alpha_{1i}] \\
& \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \wedge & & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\
& \Box_2(\neg A_4 \vee A_5 \vee A_2) \wedge & & & [\bigwedge_i \Box_2 \alpha_{2i}] \\
& \neg A_2. & & & [\gamma]
\end{array}
$$

K-SAT: Example (cont.)

$$\begin{array}{lllll}
\mu & = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\
 & & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) \wedge & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\
 & & \Box_2(\neg A_4 \vee A_5 \vee A_2) \wedge & & [\bigwedge_i \Box_2 \alpha_{2i}] \\
 & & \neg A_2. & & [\gamma]
\end{array}$$

$$\Downarrow \ \text{K-SOLVE}_{restr}()$$

$$\begin{array}{lllll}
\mu^1 & = & \Box_1(\neg A_5 \vee A_4 \vee A_3) \wedge & \Box_1(\neg A_2 \vee A_1 \vee A_4) \wedge & [\bigwedge_i \Box_1 \alpha_{1i}] \\
 & & \neg\Box_1(\neg A_3 \vee \neg A_1 \vee A_2) \wedge & \neg\Box_1(A_4 \vee \neg A_2 \vee A_3) & [\bigwedge_j \neg\Box_1 \beta_{1j}] \\
\mu^2 & = & \Box_2(\neg A_4 \vee A_5 \vee A_2) & & [\bigwedge_i \Box_2 \alpha_{2i}].
\end{array}$$

$$\Downarrow \ \text{K-SAT()}$$

$$\begin{array}{lll}
\varphi^{11} & = & (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge A_3 \wedge A_1 \wedge \neg A_2, \\
\varphi^{12} & = & (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3
\end{array}$$

K-SAT: Example (cont.)

$$\begin{array}{lll}
\varphi^{11} & = & (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge A_3 \wedge A_1 \wedge \neg A_2, \\
\varphi^{12} & = & (\neg A_5 \vee A_4 \vee A_3) \wedge (\neg A_2 \vee A_1 \vee A_4) \wedge \neg A_4 \wedge A_2 \wedge \neg A_3
\end{array}$$

$$\Downarrow \ \text{K-SOLVE()}$$

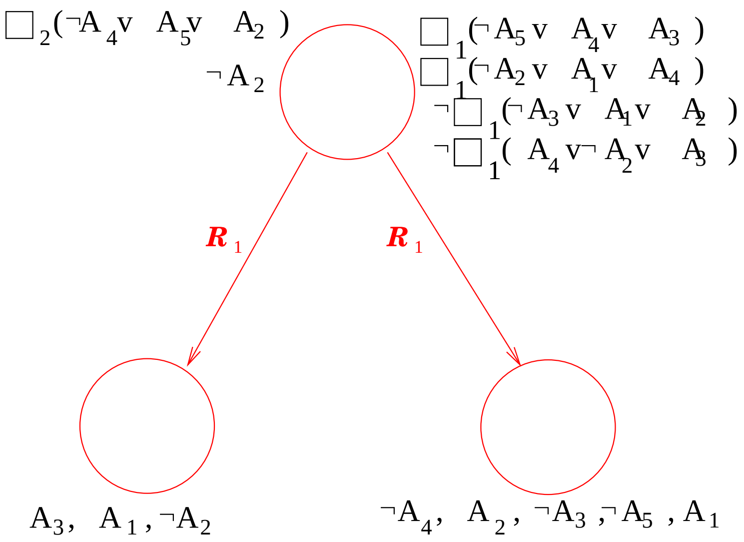
$$\begin{array}{lll}
\mu^{11} & = & A_3 \wedge A_1 \wedge \neg A_2 \\
\mu^{12} & = & \neg A_4 \wedge A_2 \wedge \neg A_3 \wedge \neg A_5 \wedge A_1
\end{array}$$

$$\Downarrow$$

$$\mathit{Satisfiable}$$

Example

Resulting Kripke Model:



$\Box_2(\neg A_4 \vee A_5 \vee A_2)$

$\neg A_2$

$\Box_1(\neg A_5 \vee A_4 \vee A_3)$

$\Box_1(\neg A_2 \vee A_1 \vee A_4)$

$\neg\Box_1(\neg A_3 \vee A_1 \vee A_2)$

$\neg\Box_1(A_4 \vee \neg A_2 \vee A_3)$

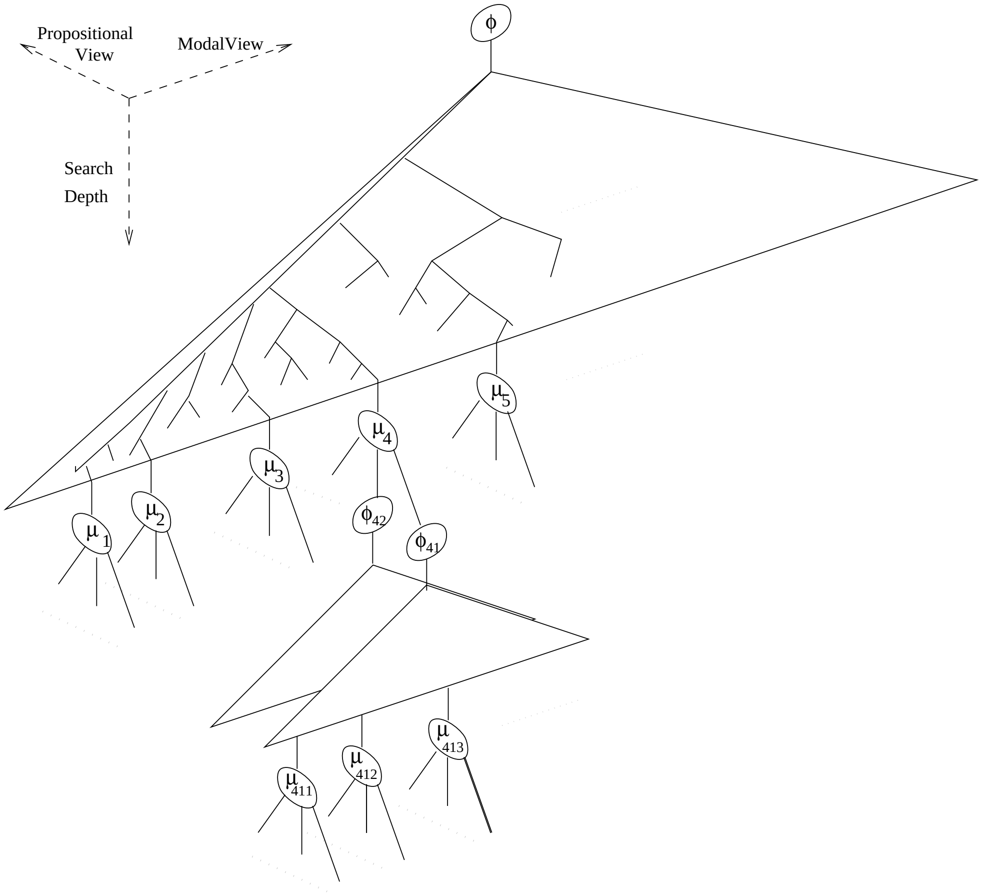
$A_3, A_1, \neg A_2$

$\neg A_4, A_2, \neg A_3, \neg A_5, A_1$

Search in modal logic:

Two alternating orthogonal components of search:

– Modal search: model spanning
  - jumping among states
  - conjunctive branching
  - up to linearly many successors
– Propositional search: local search
  - reasoning within the single states
  - disjunctive branching
  - up to exponentially many successors



Some Systems

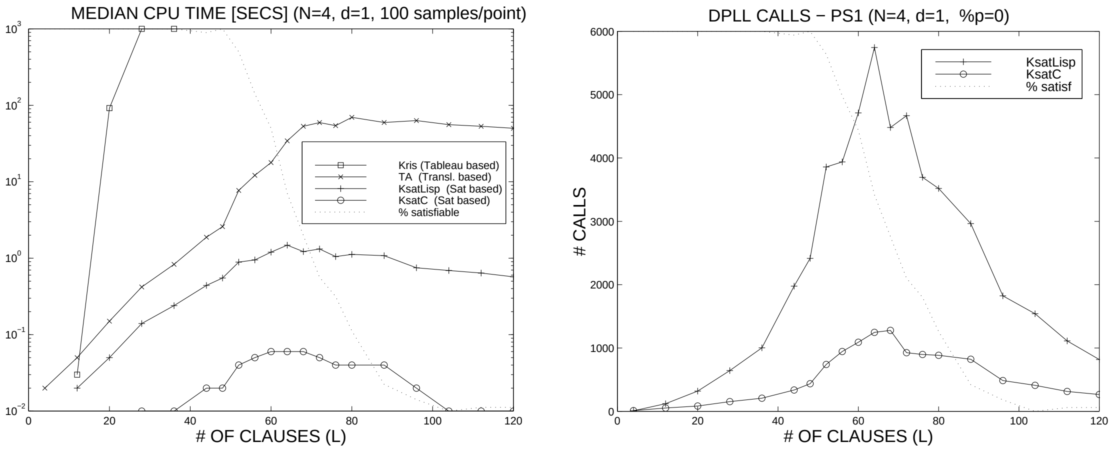
– Kris [10], CRACK [18],
  - Logics: $\mathcal{ALC}$ & many description logics
  - Boolean reasoning technique: semantic tableau
  - Optimizations: preprocessing
– K-SAT [41, 36]
  - Logics: K(m), $\mathcal{ALC}$
  - Boolean reasoning technique: DPLL
  - Optimizations: preprocessing, early pruning

Some Systems (cont.)

- FaCT & DLP [48]
  - Logics: $\mathcal{ALC}$ & many description logics
  - Boolean reasoning technique: DPLL-like
  - Optimizations: preprocessing, memoizing, backjumping + optimizations for description logics
- ESAT &*SAT [37]
  - Logics: non-normal modal logics, K(m), $\mathcal{ALC}$
  - Boolean reasoning technique: DPLL
  - Optimizations: preprocessing, early pruning, memoizing, backjumping, learning

---

Some empirical results [36]

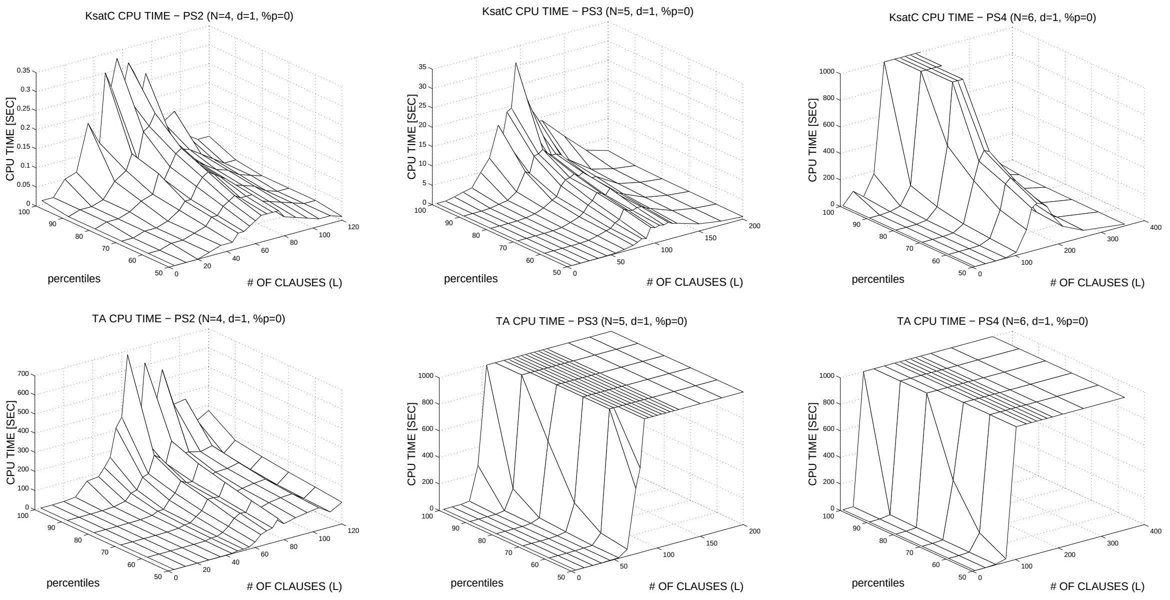


Left: KRIS, TA, K-SAT (LISP), K-SAT (C) median CPU time, 100 samples/point.

Right: K-SAT (LISP), K-SAT (C) median number of consistency checks, 100 samples/point.

Background: satisfiability percentage.

Some empirical results (cont.)



K-SAT (up) versus TA (down) CPU times.

Some empirical results [49]

Formulas of Tableau'98 competition [47]

	branch		*d4*		*dum*		*grz*		*lin*		*path*		*ph*		*poly*		*t4p*	
K	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
leanK 2.0	1	0	1	1	0	0	0	≥21	≥21	4	2	0	3	1	2	0	0	0
□KE	13	3	13	3	4	4	3	1	≥21	2	17	5	4	3	17	0	0	3
LWB 1.0	6	7	8	6	13	19	7	13	11	8	12	10	4	8	8	11	8	7
TA	9	9	≥21	18	≥21	≥21	≥21	≥21	≥21	≥21	20	20	6	9	16	17	≥21	19
***SAT 1.2**	≥21	12	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	8	12	≥21	≥21	≥21	≥21
Crack 1.0	2	1	2	3	3	≥21	1	≥21	5	2	2	6	2	3	≥21	≥21	1	1
Kris	3	3	8	6	15	≥21	13	≥21	6	9	3	11	4	5	11	≥21	7	5
Fact 1.2	6	4	≥21	8	≥21	≥21	≥21	≥21	≥21	≥21	7	6	6	7	≥21	≥21	≥21	≥21
DLP 3.1	19	13	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	≥21	7	9	≥21	≥21	≥21	≥21

	45		*branch*		*dum*		*grz*		*md*		*path*		*ph*		*poly*		*t4p*	
KT	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
TA	17	6	13	9	17	9	≥21	≥21	16	20	≥21	16	5	12	≥21	1	11	0
Kris	4	3	3	3	3	14	0	5	3	4	1	13	3	3	2	2	1	7
FaCT 1.2	≥21	≥21	6	4	11	≥21	≥21	≥21	4	5	5	3	6	7	≥21	7	4	2
DLP 3.1	≥21	≥21	19	12	≥21	≥21	≥21	≥21	3	≥21	16	14	7	≥21	≥21	12	≥21	≥21

	45		*branch*		*dum*		*grz*		*md*		*path*		*ph*		*poly*		*t4p*	
S4	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*	*p*	*n*
KT4	1	6	2	3	0	17	5	8	≥21	18	1	2	2	2	2	2	0	3
leanS4 2.0	0	0	0	0	0	0	1	1	2	2	1	0	1	0	1	1	0	0
□KE	8	0	≥21	≥21	0	≥21	6	4	3	3	9	6	4	3	1	≥21	3	1
LWB 1.0	3	5	11	7	9	≥21	8	7	8	6	8	6	4	8	4	9	9	12
TA	9	0	≥21	4	14	0	6	≥21	9	10	15	≥21	5	5	≥21	1	11	0
FaCT 1.2	≥21	≥21	4	4	2	≥21	5	4	8	4	2	1	5	4	≥21	2	5	3
DLP 3.1	≥21	≥21	18	12	≥21	≥21	10	≥21	3	≥21	15	15	7	≥21	≥21	≥21	≥21	≥21

SAT techniques for modal logics: summary

- SAT techniques have been successfully applied to modal/description logics
- Many optimizations applicable.
- Other approaches:
  - Tableaux approaches [10, 18, 46]
  - F.O. translation methods [50]
  - Inverse methods [92]
  - Automata-theoretic BDD-based methods [66, 67]

Case Study: Mathematical Reasoning

MATH-SAT [3, 95, 21, 60, 7, 6, 9, 84, 29]

- Boolean combinations of mathematical propositions on the reals or integers.
- Typically NP-complete
- Various fields of application: temporal reasoning, scheduling, formal verification, resource planning, etc.

Syntax

Let $\mathcal{D}$ be the domain of either reals $\mathbb{R}$ or integers $\mathbb{Z}$ with its set $\mathcal{OP}_{\mathcal{D}}$ of arithmetical operators.

Given a non-empty set of primitive propositions $\mathcal{A} = \{A_1, A_2, \ldots\}$ and a set $\mathcal{E}_{\mathcal{D}}$ of (linear) mathematical expressions over $\mathcal{D}$, the mathematical language $\mathcal{L}$ is the least set of formulas containing $\mathcal{A}$ and $\mathcal{E}_{\mathcal{D}}$ closed under the set of propositional connectives $\{\neg, \wedge, \vee, \rightarrow, \leftrightarrow\}$.

Syntax: math-terms and math-formulas

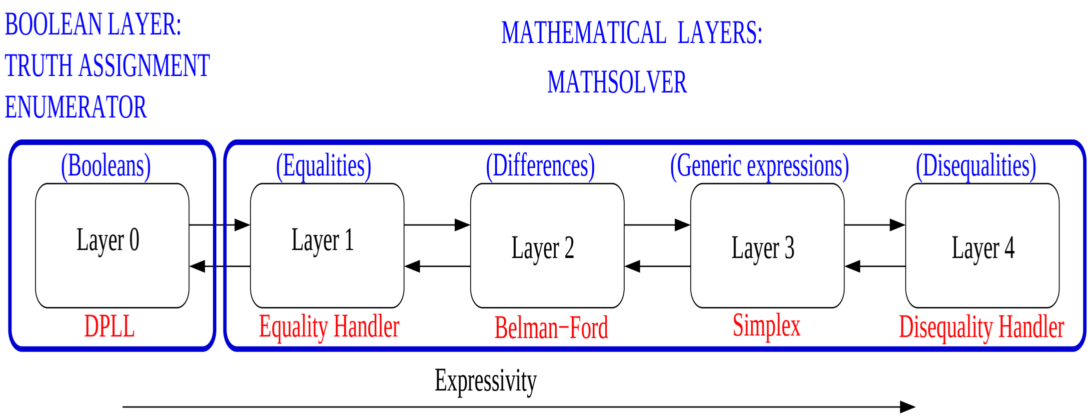
- a constant $c_i \in \mathbb{R}[\mathbb{Z}]$ is a math-term;
- a variable v_i over $\mathbb{R}[\mathbb{Z}]$ is a math-term;
- $c_i \cdot v_j$ is a math-term, $c_i \in \mathbb{R}$ and v_j being a constant and a variable over $\mathbb{R}[\mathbb{Z}]$;
- if t_1 and t_2 are math-terms, then $-t_1$ and $(t_1 \otimes t_2)$ are math-terms, $\otimes \in \{+, -\}$.
- a boolean proposition A_i over $\mathbb{B} := \{\bot, \top\}$ is a math-formula;
- if t_1, t_2 are math-terms, then $(t_1 \bowtie t_2)$ is a math-formula, $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$;
- if φ_1, φ_2 are math-formulas, then $\neg\varphi_1$, $(\varphi_1 \wedge \varphi_2)$, $(\varphi_1 \vee \varphi_2)$, $(\varphi_1 \rightarrow \varphi_2)$ and $(\varphi_1 \leftrightarrow \varphi_2)$, are math-formulas.

Interpretations

Interpretation: a map I assigning real [integer] and boolean values to math-terms and math-formulas respectively and preserving constants and operators:

- $I(A_i) \in \{\top, \bot\}$, for every $A_i \in \mathcal{A}$;
- $I(c_i) = c_i$, for every constant $c_i \in \mathbb{R}$;
- $I(v_i) \in \mathbb{R}$, for every variable v_i over $\mathbb{R}$;
- $I(t_1 \otimes t_2) = I(t_1) \otimes I(t_2)$, for all math-terms t_1, t_2 and $\otimes \in \{+, -, \cdot\}$;
- $I(t_1 \bowtie t_2) = I(t_1) \bowtie I(t_2)$, for all math-terms t_1, t_2 and $\bowtie \in \{=, \neq, >, <, \geq, \leq\}$;
- $I(\neg\varphi_1) = \neg I(\varphi_1)$, for every math-formula φ_1;
- $I(\varphi_1 \wedge \varphi_2) = I(\varphi_1) \wedge I(\varphi_2)$, for all math-formulas φ_1, φ_2.

MATH-SAT: A Layered Architecture [6]



- ▷ Organized in layers of increasing expressive power,
- ▷ Specialized algorithms for particular propositions
- ▷ Each layer comes into play only when needed

Motivating application domains

- Propositional bounded model checking (BMC) [15]: layer 0
  purely propositional atoms $A_1, A_2, \ldots$
- BMC for timed system (no-loops) [9]: layers 0-2
  atoms in the form $A_i, (x = y), (x - y \leq C)$
- BMC for timed (with-loops) and hybrid systems [9, 8]: layers 0-3
  atoms in the form $A_i, (x = y), (x - y \leq C), (x - y = z - w)$
- ...

Layer 0: modified DPLL (SIM)

```
bool MATH-SAT($\varphi, \mu$)
    if ($\varphi == \top$)                                  /* base         */
        then return (MATH-SOLVE($\mu$)==satisfiable);
    if ($\varphi == \bot$)                                  /* backtrack */
        then return False;
    if {a unit clause $(l)$ occurs in $\varphi$}            /* unit        */
        then return MATH-SAT($assign(l, \varphi), \mu \wedge l$);
    if Likely-Unsatisfiable($\mu$)                          /* early pruning */
        if (MATH-SOLVE($\mu$) == False)
            then return False;
    l = choose-literal($\varphi$);                          /* split        */
    return    MATH-SAT($assign(l, \varphi), \mu \wedge l$)   or
              MATH-SAT($assign(\neg l, \varphi), \mu \wedge \neg l$);
```

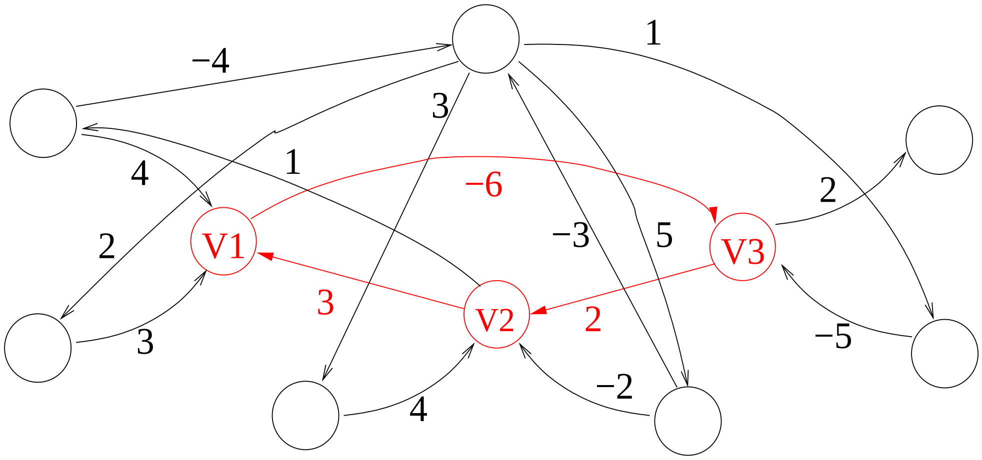
Layer 1: Eliminating Equalities

1. Reveal equalities. Build equivalence classes.
   E.g.: $\{\underline{(v_i = v_j)}, \underline{(v_j = v_k)}, (v_i - v_j \leq 3), (v_i - v_k \leq -2), \ldots\}$
2. Eliminate equivalences and substitute variables:
   $\Longrightarrow \{\ldots, (v_k - v_k \leq 3), (v_k - v_k \leq -2), \ldots\}$
3. Remove all valid atoms, reveal inconsistent atoms:
   - Return "false" if there are inconsistent atoms.
     $\{\ldots, \underline{(v_k - v_k \leq -2)}, \ldots\} \Longrightarrow$ false
   - Invoke layer 2 on the resulting set otherwise.

Layer 2: Handling differences

▷ Deals with "difference" atoms: $(x - y \leq C)$
$\{\ldots, (v_1 - v_2 \leq 3), (v_2 - v_3 \leq 2), (v_3 - v_1 \leq -6), \ldots\}$

▷ Use Belman-Ford's minimal path algorithm with negative cycle detection



Layer 3: dealing with other linear expressions

E.g., $\{(x - y = z - w), (2x - 3y + 4z \leq 5), \ldots\}$

- ▷ Invoke a Simplex Algorithm

Layer 4: Dealing with disequalities

- ▷ Unnecessary with the problems of our interest
  (BMC for timed & hybrid systems)
- ▷ Lazy approach:
  1. Use levels 1, 2, 3 ignoring disequalities. If unsatisfiable, return false.
  2. If the interpretation found verifies the disequalities, return it.
  3. Otherwise, split the two subcases and restart
     $(x \neq y) \Longrightarrow (x < y) \vee (x > y)$
- ▷ Alternative: expand into a disjunction, add mutex clauses

$$\begin{array}{lll} \ldots \vee (x < y) \vee (x > y) & \wedge & instead\ of\ \text{“}\ldots \vee \neg(x = y)\text{''} \\ \neg(x < y) \vee \neg(x > y) & \wedge & \\ \neg(x = y) \vee \neg(x < y) & \wedge & iff\ (x = y)\ occurs\ positively \\ \neg(x = y) \vee \neg(x > y) & & \quad '' \quad '' \quad '' \quad '' \end{array}$$

Some Systems

- Tsat [3]
  - Logics: disjunctions of difference expressions (positive math-atoms only)
  - Applications: temporal reasoning
  - Boolean reasoning technique: DPLL
  - Optimizations: preprocessing, static learning, forward checking
- LPsat [95]
  - Logics: MATH-SAT (positive math-atoms only)
  - Applications: resource planning
  - Boolean reasoning technique: DPLL
  - Optimizations: preprocessing, backjumping, learning, triggering

---

Some systems (cont.)

- DDD [60]
  - Logics: boolean + difference expressions
  - Applications: formal verification of timed systems
  - Boolean reasoning technique: OBDD
  - Optimizations: preprocessing, early pruning
- MATH-SAT [6]
  - Logics: MATH-SAT
  - Applications: resource planning, formal verification of timed systems
  - Boolean reasoning technique: DPLL
  - Optimizations: preprocessing, enhanced early pruning, backjumping, learning, triggering

Some systems (cont.)

- CVC [12, 89]
  - Logics: boolean + linear real arithmetic + arrays + inductive datatypes
  - Applications: formal verification
  - Boolean reasoning technique: DPLL
  - Optimizations: backjumping,learning
- ICS [78, 31]
  - Logics: boolean + linear real arithmetic + arrays
  - Applications: formal verification
  - Boolean reasoning technique: DPLL
  - Optimizations: backjumping,learning
- ...

Related systems

Other related systems:

- RDL [4]
- Simplify [64]
- STeP [16]
- UCLID [77]
- ...

SAT + mathematical reasoning: summary

- SAT techniques have been successfully applied to MATH-SAT
- Many optimizations applicable.
- Currently competitive with state-of-the-art applications for temporal reasoning, resource planning, formal verification of timed systems, formal verification of circuits at abstract level.

References

[1] P. A. Abdullah, P. Bjesse, and N. Een. Symbolic Reachability Analysis based on SAT-Solvers. In *Sixth Int.nl Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS’00)*, 2000.

[2] A. Armando. Simplifying OBDDs in Decidable Theories. In *Proc. of the 1st CADE-19 Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR’03)*, 2003.

[3] A. Armando, C. Castellini, and E. Giunchiglia. SAT-based procedures for temporal reasoning. In *Proc. European Conference on Planning, CP-99*, 1999.

[4] A Armando, L. Compagna, and S. Ranise. System Description: RDL—Rewrite and Decision procedure Laboratory. In *Proc IJCAR 01*, LNAI. Springer, 2001.

[5] A. Armando and E. Giunchiglia. Embedding Complex Decision Procedures inside an Interactive Theorem Prover. *Annals of Mathematics and Artificial Intelligence*, 8(3–4):475–502, 1993.

[6] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. A SAT Based Approach for Solving Formulas over Boolean and Linear Mathematical Propositions. In *Proc. CADE’2002.*, volume 2392 of *LNAI*. Springer, July 2002.

[7] G. Audemard, P. Bertoli, A. Cimatti, A. Korniłowicz, and R. Sebastiani. Integrating Boolean and Mathematical Solving: Foundations, Basic Algorithms and Requirements. In *Artificial*

Intelligence, Automated Reasoning, and Symbolic Computation, Proc. of the Joint Int.nal Conference, volume 2385 of *LNAI*. Springer, 2002.

[8] G. Audemard, M. Bozzano, A. Cimatti, and R. Sebastiani. Verifying Industrial Hybrid Systems with MathSAT. In *Proc. of the 1st CADE-19 Workshop on Pragmatics of Decision Procedures in Automated Reasoning (PDPAR'03)*, 2003.

[9] G. Audemard, A. Cimatti, A. Korniłowicz, and R. Sebastiani. SAT-Based Bounded Model Checking for Timed Systems. In *Proc. FORTE'02.*, volume 2529 of *LNCS*. Springer, November 2002.

[10] F. Baader, E. Franconi, B. Hollunder, B. Nebel, and H.J. Profitlich. An Empirical Analysis of Optimization Techniques for Terminological Representation Systems or: Making KRIS get a move on. *Applied Artificial Intelligence. Special Issue on Knowledge Base Management*, 4:109–132, 1994.

[11] F. Bacchus and J. Winter. Effective Preprocessing with Hyper-Resolution and Equality Reduction. In *Proc. Sixth International Symposium on Theory and Applications of Satisfiability Testing*, 2003.

[12] C. Barrett, D. Dill, and A. Stump. Checking Satisfiability of First-Order Formulas by Incremental Translation to SAT. In *14th International Conference on Computer-Aided Verification*, 2002.

[13] Clark W. Barrett, David L. Dill, and Aaron Stump. A generalization of Shostak's method for combining decision procedures. In *Frontiers of Combining Systems (FROCOS)*, Lecture Notes in Artificial Intelligence. Springer-Verlag, April 2002. Santa Margherita Ligure, Italy.

---

[14] R. J. Bayardo, Jr. and R. C. Schrag. Using CSP Look-Back Techniques to Solve Real-World SAT instances. In *American Association for Artificial Intelligence*, pages 203–208. AAAI Press, 1997.

[15] A. Biere, A. Cimatti, E. M. Clarke, and Yunshan Zhu. Symbolic Model Checking without BDDs. In *Proc. TACAS'99*, pages 193–207, 1999.

[16] N. Bjorner, Z. Manna, H. Sipma, and T. Uribe. Deductive Verification of Real-time systems using STeP. *Theoretical Computer Science*, 253, 2001.

[17] R. Brafman. A simplifier for propositional formulas with many binary clauses. In *Proc. IJCAI01*, 2001.

[18] P. Bresciani, E. Franconi, and S. Tessaris. Implementing and testing expressive Description Logics: a preliminary report. In *Proc. International Workshop on Description Logics*, Rome, Italy, 1995.

[19] R. E. Bryant. Graph-Based Algorithms for Boolean Function Manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[20] G. Cabodi, P. Camurati, and S. Quer. Can BDDs compete with SAT solvers on Bounded Model Checking? In *Proc. DAC'39: 39st ACM/IEEE Design Automation Conference*, New Orleans, Louisiana, June 2002.

[21] W. Chan, R. J. Anderson, P. Beame, and D. Notkin. Combining constraint solving and symbolic model checking for a class of systems with non-linear constraints. In *Proc. CAV'97*, volume 1254 of *LNCS*, pages 316–327, Haifa, Israel, June 1997. Springer-Verlag.

[22] A. Cimatti, E. Clarke, E. Giunchiglia, F. Giunchiglia, M. Pistore, M. Roveri, R. Sebastiani, and A. Tacchella. NuSMV Version 2: An OpenSource Tool for Symbolic Model Checking. In *Proc. International Conference on Computer-Aided Verification (CAV 2002)*, volume 2404 of *LNCS*, Copenhagen, Denmark, July 2002. Springer.

[23] A. Cimatti, M. Pistore, M. Roveri, and R. Sebastiani. Improving the Encoding of LTL Model Checking into SAT. In *Proc. VMCAI'02*, volume 2294 of *LNCS*. Springer, January 2002.

[24] S. A. Cook. The complexity of theorem proving procedures. In *3rd Annual ACM Symposium on the Theory of Computation*, pages 151–158, 1971.

[25] M. D'Agostino and M. Mondadori. The Taming of the Cut. *Journal of Logic and Computation*, 4(3):285–319, 1994.

[26] M. Davis, G. Longemann, and D. Loveland. A machine program for theorem proving. *Journal of the ACM*, 5(7), 1962.

[27] M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, 7:201–215, 1960.

[28] T. Boy de la Tour. Minimizing the Number of Clauses by Renaming. In *Proc. of the 10th Conference on Automated Deduction*, pages 558–572. Springer-Verlag, 1990.

[29] L. de Moura, H. Ruess, and M. Sorea. Lazy Theorem Proving for Bounded Model Checking over Infinite Domains. In *Proc. CADE'2002.*, volume 2392 of *LNAI*. Springer, July 2002.

[30] M. Ernst, T. Millstein, and D. Weld. Automatic SAT-compilation of planning problems. In

Proc. IJCAI-97, 1997.

[31] Jean-Christophe Filliâtre, Sam Owre, Harald Rueß, and N. Shankar. Ics: Integrated canonizer and solver. Proc. CAV'2001, 2001.

[32] M. Fitting. First-Order Modal Tableaux. *Journal of Automated Reasoning*, 4:191–213, 1988.

[33] M. R. Garey and D. S. Johnson. *Computers and Intractability*. Freeman and Company, New York, 1979.

[34] A. Van Gelder. A satisfiability tester for non-clausal propositional calculus. *Information and Computation*, 79:1–21, October 1988.

[35] I. P. Gent, E. MacIntyre, P. Prosser, and T. Walsh. The constrainedness of search. In *Proceedings of AAAI-96*, pages 246–252, Menlo Park, 1996. AAAI Press / MIT Press.

[36] E. Giunchiglia, F. Giunchiglia, R. Sebastiani, and A. Tacchella. SAT vs. Translation based decision procedures for modal logics: a comparative evaluation. *Journal of Applied Non-Classical Logics*, 10(2):145–172, 2000.

[37] E. Giunchiglia, F. Giunchiglia, and A. Tacchella. SAT Based Decision Procedures for Classical Modal Logics. Journal of Automated Reasoning. Special Issue: Satisfiability at the start of the year 2000, 2001.

[38] E. Giunchiglia, A. Massarotto, and R. Sebastiani. Act, and the Rest Will Follow: Exploiting Determinism in Planning as Satisfiability. In *Proc. AAAI'98*, pages 948–953, 1998.

[39] E. Giunchiglia, M. Narizzano, A. Tacchella, and M. Vardi. Towards an Efficient Library for

SAT: a Manifesto. In *Proc. SAT 2001*, Electronics Notes in Discrete Mathematics. Elsevier Science., 2001.

[40] E. Giunchiglia and R. Sebastiani. Applying the Davis-Putnam procedure to non-clausal formulas. In *Proc. AI*IA'99*, volume 1792 of *LNAI*. Springer, 1999.

[41] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K. In *Proc. CADE'13*, LNAI, New Brunswick, NJ, USA, August 1996. Springer.

[42] F. Giunchiglia and R. Sebastiani. A SAT-based decision procedure for ALC. In *Proc. of the 5th International Conference on Principles of Knowledge Representation and Reasoning - KR'96*, Cambridge, MA, USA, November 1996.

[43] F. Giunchiglia and R. Sebastiani. Building decision procedures for modal logics from propositional decision procedures - the case study of modal K(m). *Information and Computation*, 162(1/2), October/November 2000.

[44] J. Y. Halpern. The effect of bounding the number of primitive propositions and the depth of nesting on the complexity of modal logic. *Artificial Intelligence*, 75(3):361–372, 1995.

[45] J.Y. Halpern and Y. Moses. A guide to the completeness and complexity for modal logics of knowledge and belief. *Artificial Intelligence*, 54(3):319–379, 1992.

[46] J. Happe. The ModProf Theorem Prover. In *Proc IJCAR 01*, LNAI. Springer, 2001.

[47] A. Heuerding and S. Schwendimann. A benchmark method for the propositional modal logics K, KT, S4. Technical Report IAM-96-015, University of Bern, Switzerland, 1996.

[48] I. Horrocks and P. F. Patel-Schneider. FaCT and DLP. In *Proc. Tableaux'98*, pages 27–30, 1998.

[49] I. Horrocks, P. F. Patel-Schneider, and R. Sebastiani. An Analysis of Empirical Testing for Modal Decision Procedures. *Logic Journal of the IGPL*, 8(3):293–323, May 2000.

[50] U. Hustadt and R.A. Schmidt. An empirical analysis of modal theorem provers. *Journal of Applied Non-Classical Logics*, 9(4), 1999.

[51] H. Kautz, D. McAllester, and B. Selman. Encoding Plans in Propositional Logic. In *Proceedings International Conference on Knowledge Representation and Reasoning*. AAAI Press, 1996.

[52] H. Kautz and B. Selman. Planning as Satisfiability. In *Proc. ECAI-92*, pages 359–363, 1992.

[53] S. Kirkpatrick and B. Selman. Critical behaviour in the satisfiability of random boolean expressions. *Science*, 264:1297–1301, 1994.

[54] S. A. Kripke. Semantical considerations on modal logic. In *Proc. A colloquium on Modal and Many-Valued Logics*, Helsinki, 1962.

[55] Chu Min Li and Anbulagan. Heuristics based on unit propagation for satisfiability problems. In *Proceedings of the 15th International Joint Conference on Artificial Intelligence (IJCAI-97)*, pages 366–371, 1997.

[56] M. Mahfoudh, P. Niebert, E. Asarin, and O. Maler. A Satisfibaility Checker for Difference Logic. In *Proceedings of SAT-02*, pages 222–230, 2002.

[57] F. Massacci and L. Marraro. Logical Cryptanalysis as a SAT Problem. *Journal of Automated Reasoning*, 24(1/2):165–203, 2000.

[58] K. McMillan. Applying SAT Methods in Unbounded Symbolic Model Checking. In *Proc. CAV*. Springer, 2002.

[59] D. Mitchell, B. Selman, and H. Levesque. Hard and Easy Distributions of SAT Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 459–465, 1992.

[60] J. Moeller, J. Lichtenberg, H. R. Andersen, and H. Hulgaard. Fully symbolic model checking of timed systems using difference decision diagrams. In *Proc. Workshop on Symbolic Model Checking (SMC), Federated Logic Conference (FLoC)*, Trento, Italy, July 1999.

[61] M. W. Moskewicz, C. F. Madigan, Y. Z., L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *Design Automation Conference*, 2001.

[62] C. G. Nelson and D. C. Oppen. Simplification by cooperating decision procedures. *TOPLAS*, 1(2):245–257, 1979.

[63] G. Nelson and D.C. Oppen. Fast Decision Procedures Based on Congruence Closure. *Journal of the ACM*, 27(2):356–364, 1980.

[64] Greg Nelson. Combining satisfiability procedures by equality-sharing. In W. W. Bledsoe and D. W. Loveland, editors, *Automated Theorem Proving: After 25 Years*, volume 29 of *Contemporary Mathematics*, pages 201–211. American Mathematical Society, Providence, RI, 1984.

[65] P. Niebert, M. Mahfoudh, E. Asarin, M. Bozga, and O. Maler. Verification of Timed Automata via Satisfiability Checking. In *Proc. of FTRTFT'02*, LNCS. Springer-Verlag, 2002.

[66] G. Pan, U. Sattler, and M. Y. Vardi. BDD-Based Decision Procedures for K. In *Proc. CADE*, LNAI. Springer, 2002.

[67] G. Pan and M. Y. Vardi. Optimizing a BDD-based modal solver. In *Proc. CADE*, LNAI. Springer, 2003.

[68] P. F. Patel-Schneider and I. Horrocks. DLP and FaCT. In *Proc. Tableaux'99*, pages 19–23, 1999.

[69] P. F. Patel-Schneider and R. Sebastiani. A New General Method to Generate Random Modal Formulae for Testing Decision Procedures. *Journal of Artificial Intelligence Research, (JAIR)*, 18:351–389, May 2003. Morgan Kaufmann.

[70] W. Penczek, B. Woźna, and A. Zbrzezny. Towards bounded model checking for the universal fragment of TCTL. In *Proc. of FTRTFT'02*, LNCS. Springer-Verlag, 2002.

[71] D.A. Plaisted and S. Greenbaum. A Structure-preserving Clause Form Translation. *Journal of Symbolic Computation*, 2:293–304, 1986.

[72] K. D. Schild. A correspondence theory for terminological logics: preliminary report. In *Proc. of the 12th International Joint Conference on Artificial Intelligence*, pages 466–471, Sydney, Australia, 1991.

[73] R. Sebastiani. Applying GSAT to Non-Clausal Formulas. *Journal of Artificial Intelligence Research*, 1:309–314, 1994.

[74] B. Selman and H. Kautz. Domain-Independent Extension to GSAT: Solving Large Structured Satisfiability Problems. In *Proc. of the 13th International Joint Conference on Artificial Intelligence*, pages 290–295, 1993.

[75] B. Selman, H. Kautz, and B. Cohen. Local Search Strategies for Satisfiability Testing. In *Cliques, Coloring, and Satisfiability*, volume 26 of *DIMACS*, pages 521–532, 1996.

[76] B. Selman, H. Levesque., and D. Mitchell. A New Method for Solving Hard Satisfiability Problems. In *Proc. of the 10th National Conference on Artificial Intelligence*, pages 440–446, 1992.

[77] S. A. Seshia, S. K. Lahiri, and R. E. Bryant. A Hybrid SAT-Based Decision Procedure for Separation Logic with Uninterpreted Functions. In *Proc. 40th Design Automation Conference (DAC)*, 2003.

[78] N. Shankar and Harald Rueß. Combining shostak theories. Invited paper for Floc'02/RTA'02, 2002.

[79] M. Sheeran and G. Stalmarck. A tutorial on Stalmarck's proof procedure. In *Proc. FMCAD*, 1998.

[80] D. Sheridan and T. Walsh. Clause Forms Generated by Bounded Model Checking. In *Proc. Eighth Workshop on Automated Reasoning: Bridging the Gap between Theory and Practice*, University of York, March 2001.

[81] R. Shostak. A Pratical Decision Procedure for Arithmetic with Function Symbols. *Journal of the ACM*, 26(2):351–360, 1979.

[82] J. P. M. Silva and K. A. Sakallah. GRASP - A new Search Algorithm for Satisfiability. In *Proc. ICCAD'96*, 1996.

[83] R. M. Smullyan. *First-Order Logic*. Springer-Verlag, NY, 1968.

[84] Maria Sorea. Bounded model checking for timed automata. *ENTCS*, 68(5), 2002.

[85] P. Stephan, R. Brayton, , and A. Sangiovanni-Vencentelli. Combinational Test Generation Using Satisfiability. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15:1167–1176, 1996.

[86] O. Strichman. On Solving Presburger and Linear Arithmetic with SAT. In *Proc. of Formal Methods in Computer-Aided Design (FMCAD 2002)*, LNCS. Springer, 2002.

[87] O. Strichman, S. Seshia, and R. Bryant. Deciding separation formulas with SAT. In *Proc. of Computer Aided Verification, (CAV'02)*, LNCS. Springer, 2002.

[88] O. Strichmann. Tuning SAT checkers for Bounded Model Checking. In *Proc. CAV00*, volume 1855 of *LNCS*, pages 480–494. Springer, 2000.

[89] A. Stump, C. W. Barrett, and D. L. Dill. CVC: A Cooperating Validity Checker. In *Proc. CAV'02*, number 2404 in LNCS. Springer Verlag, 2002.

[90] C. Tinelli. A DPLL-based Calculus for Ground Satisfiability Modulo Theories. In G. Ianni and S. Flesca, editors, *Proceedings of the 8th European Conference on Logics in Artificial Intelligence (Cosenza, Italy)*, volume 2424 of *LNAI*, pages 308–319. Springer, 2002.

[91] R. Moeller V. Haarslev. RACER System Description. In *Proc. of International Joint*

Conference on Automated reasoning - IJCAR-2001, volume 2083 of *LNAI*, Siena, Italy, July 2001. Springer-verlag.

[92] A Voronkov. How to Optimize Proof-Search in Modal Logics: A New Way of Proving Redundancy Criteria for Sequent Calculi. In *Proc. LICS'00*, June 2000.

[93] C. P. Williams and T. Hogg. Exploiting the deep structure of constraint problems. *Artificial Intelligence*, 70:73–117, 1994.

[94] P. F. Williams, A. Biere, E. M. Clarke, and A. Gupta. Combining Decision Diagrams and SAT Procedures for Efficient Symbolic Model Checking. In *Proc. CAV2000*, volume 1855 of *LNCS*, pages 124–138, Berlin, 2000. Springer.

[95] S. Wolfman and D. Weld. The LPSAT Engine & its Application to Resource Planning. In *Proc. IJCAI*, 1999.

[96] H. Zhang and M. Stickel. Implementing the Davis-Putnam algorithm by tries. Technical report, University of Iowa, August 1994.

[97] Lintao Zhang and Sharad Malik. The quest for efficient boolean satisfiability solvers. In *Proc. CAV'02*, number 2404 in LNCS, pages 17–36. Springer, 2002.

DISCLAIMER
The list of references above is by no means intended to be all-inclusive. The author of these slides apologizes both with the authors and with the readers for all the relevant works which are not cited here.

The papers (co)authored by the author of these slides are availlable at:
`http://www.dit.unitn.it/~rseba/publist.html`.

Related web sites:

- Combination Methods in Automated Reasoning
  `http://combination.cs.uiowa.edu/`
- SMT-LIB - The Satisfiability Modulo Theories Library
  `http://goedel.cs.uiowa.edu/smtlib/`
- SATLive! - Up-to-date links for SAT
  `http://www.satlive.org/index.jsp`
- SATLIB - The Satisfiability Library
  `http://www.intellektik.informatik.tu-darmstadt.de/SATLIB/`