Decision Procedures

Jochen Hoenicke

Software Engineering
Albert-Ludwigs-University Freiburg

Summer 2012

Organisation

Organisation

Dates

- Lecture is Tuesday 14–16 (c.t) and Thursday 14–15 (c.t).
- Tutorials will be given on Thursday 15–16.
  Starting next week (this week is a two hour lecture).
- Exercise sheets are uploaded on Tuesday.
  They are due on Tuesday the week after.

To successfully participate, you must

- prepare the exercises (at least 50 %)
- actively participate in the tutorial
- pass an oral examination

The Calculus of Computation:
Decision Procedures with
Applications to Verification

by
Aaron Bradley
Zohar Manna

Springer 2007

Motivation

Motivation

Decision Procedures are algorithms to decide formulae.

These formulae can arise

- in Hoare-style software verification.
- in hardware verification

Consider the following program:

```
for
  @ ℓ ≤ i ≤ u ∧ (rv ↔ ∃j. ℓ ≤ j < i ∧ a[j] = e)
  (int i := ℓ; i ≤ u; i := i + 1) {
  if ((a[i] = e)) {
    rv := true;
  }
}
```

How can we prove that the formula is a loop invariant?

Motivation (3)

Prove the Hoare triples (one for if case, one for else case)

$\texttt{assume}\ \ell \leq i \leq u \land (rv \leftrightarrow \exists j.\ \ell \leq j < i \land a[j] = e)$
$\texttt{assume}\ i \leq u$
$\texttt{assume}\ a[i] = e$
$rv := \texttt{true};$
$i := i + 1$
$@\ \ell \leq i \leq u \land (rv \leftrightarrow \exists j.\ \ell \leq j < i \land a[j] = e)$

$\texttt{assume}\ \ell \leq i \leq u \land (rv \leftrightarrow \exists j.\ \ell \leq j < i \land a[j] = e)$
$\texttt{assume}\ i \leq u$
$\texttt{assume}\ a[i] \neq e$
$i := i + 1$
$@\ \ell \leq i \leq u \land (rv \leftrightarrow \exists j.\ \ell \leq j < i \land a[j] = e)$

A Hoare triple {P} S {Q} holds, iff

$$P \rightarrow wp(S, Q)$$

(wp denotes is weakest precondition)
For assignments wp is computed by substitution:

---

assume $\ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$
assume $i \leq u$
assume $a[i] = e$
$rv :=$ true;
$i := i + 1$
@ $\ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e)$

---

holds if and only if:

$$\begin{aligned}&\ell \leq i \leq u \wedge (rv \leftrightarrow \exists j.\ \ell \leq j < i \wedge a[j] = e) \wedge i \leq u \wedge a[i] = e\\ \rightarrow &\ell \leq i + 1 \leq u \wedge (\texttt{true} \leftrightarrow \exists j.\ \ell \leq j < i + 1 \wedge a[j] = e)\end{aligned}$$

Motivation (5)

We need an algorithm that decides whether a formula holds.

$$\begin{aligned} &\ell \leq i \leq u \land (rv \leftrightarrow \exists j.\, \ell \leq j < i \land a[j] = e) \land i \leq u \land a[i] = e \\ \rightarrow &\ell \leq i + 1 \leq u \land (\texttt{true} \leftrightarrow \exists j.\, \ell \leq j < i + 1 \land a[j] = e) \end{aligned}$$

If the formula does not hold it should give a counterexample, e.g.:

$$\ell = 0, i = 1, u = 1, rv = \mathit{false}, a[0] = 0, a[1] = 1, e = 1,$$

This counterexample shows that $i + 1 \leq u$ can be violated.

This lecture is about algorithms checking for validity and producing these counterexamples.

Contents of Lecture

Topics

- Propositional Logic
- First-Order Logic
- First-Order Theories
- Quantifier Elimination
- Decision Procedures for Linear Arithmetic
- Decision Procedures for Uninterpreted Functions
- Decision Procedures for Arrays
- Combination of Decision Procedures
- DPLL(T)
- Craig Interpolants

Foundations: Propositional Logic

Syntax of Propositional Logic

Atom	truth symbols $\top$ ("true") and $\bot$ ("false")
	propositional variables $P, Q, R, P_1, Q_1, R_1, \cdots$
Literal	atom α or its negation $\neg\alpha$
Formula	literal or application of a logical connective to formulae F, F_1, F_2

$\neg F$	"not"	(negation)
$(F_1 \wedge F_2)$	"and"	(conjunction)
$(F_1 \vee F_2)$	"or"	(disjunction)
$(F_1 \rightarrow F_2)$	"implies"	(implication)
$(F_1 \leftrightarrow F_2)$	"if and only if"	(iff)

Example: Syntax

formula F : $((P \wedge Q) \rightarrow (\top \vee \neg Q))$
atoms: $P, Q, \top$
literal: $\neg Q$
subformulas: $(P \wedge Q), \quad (\top \vee \neg Q)$
abbreviation

$$F : P \wedge Q \rightarrow \top \vee \neg Q$$

Formula F and Interpretation I is evaluated to a truth value $0/1$

where 0 corresponds to value false
1 true

Interpretation I : $\{P \mapsto 1, Q \mapsto 0, \cdots\}$

Evaluation of logical operators:

F_1	F_2	$\neg F_1$	$F_1 \wedge F_2$	$F_1 \vee F_2$	$F_1 \rightarrow F_2$	$F_1 \leftrightarrow F_2$
0	0	1	0	0	1	1
0	1		0	1	1	0
1	0	0	0	1	0	0
1	1		1	1	1	1

Example: Semantics

$F : P \land Q \rightarrow P \lor \neg Q$
$I : \{P \mapsto 1, Q \mapsto 0\}$

P	Q	$\neg Q$	$P \land Q$	$P \lor \neg Q$	F
1	0	1	0	1	1

$1 =$ true $\qquad 0 =$ false

F evaluates to true under I

Inductive Definition of PL's Semantics

$I \models F$ if F evaluates to 1 / true under I

$I \not\models F$ 0 / false

Base Case:

$I \models \top$

$I \not\models \bot$

$I \models P$ iff $I[P] = 1$

$I \not\models P$ iff $I[P] = 0$

Inductive Case:

$I \models \neg F$ iff $I \not\models F$

$I \models F_1 \wedge F_2$ iff $I \models F_1$ and $I \models F_2$

$I \models F_1 \vee F_2$ iff $I \models F_1$ or $I \models F_2$

$I \models F_1 \rightarrow F_2$ iff, if $I \models F_1$ then $I \models F_2$

$I \models F_1 \leftrightarrow F_2$ iff, $I \models F_1$ and $I \models F_2$,
or $I \not\models F_1$ and $I \not\models F_2$

Example: Inductive Reasoning

$$F : P \wedge Q \rightarrow P \vee \neg Q$$

$$I : \{P \mapsto 1,\ Q \mapsto 0\}$$

1. $I \models P$	since $I[P] = 1$	
2. $I \not\models Q$	since $I[Q] = 0$	
3. $I \models \neg Q$	by 2, $\neg$	
4. $I \not\models P \wedge Q$	by 2, $\wedge$	
5. $I \models P \vee \neg Q$	by 1, $\vee$	
6. $I \models F$	by 4, $\rightarrow$	Why?

Thus, F is true under I.

Satisfiability and Validity

UNI FREIBURG

Definition (Satisfiability)

F is satisfiable iff there exists an interpretation I such that $I \models F$.

Definition (Validity)

F is valid iff for all interpretations I, $I \models F$.

Note

F is valid iff $\neg F$ is unsatisfiable

Proof.

F is valid iff $\forall I : I \models F$ iff $\neg\exists I : I \not\models F$ iff $\neg F$ is unsatisfiable. □

Decision Procedure: An algorithm for deciding validity or satisfiability.

Examples: Satisfiability and Validity

Now assume, you are a decision procedure.

Which of the following formulae is satisfiable, which is valid?

- $F_1 : P \land Q$
  satisfiable, not valid
- $F_2 : \neg(P \land Q)$
  satisfiable, not valid
- $F_3 : P \lor \neg P$
  satisfiable, valid
- $F_4 : \neg(P \lor \neg P)$
  unsatisfiable, not valid
- $F_5 : (P \rightarrow Q) \land (P \lor Q) \land \neg Q$
  unsatisfiable, not valid

Is there a formula that is unsatisfiable and valid?

Method 1: Truth Tables

$F : P \wedge Q \rightarrow P \vee \neg Q$

P	Q	$P \wedge Q$	$\neg Q$	$P \vee \neg Q$	F
0	0	0	1	1	1
0	1	0	0	0	1
1	0	0	1	1	1
1	1	1	0	1	1

Thus F is valid.

$F : P \vee Q \rightarrow P \wedge Q$

P	Q	$P \vee Q$	$P \wedge Q$	F	
0	0	0	0	1	$\leftarrow$ satisfying I
0	1	1	0	0	$\leftarrow$ falsifying I
1	0	1	0	0	
1	1	1	1	1	

Thus F is satisfiable, but invalid.

Method 2: Semantic Argument (Semantic Tableaux)

- Assume F is not valid and I a falsifying interpretation: $I \not\models F$
- Apply proof rules.
- If no contradiction reached and no more rules applicable, F is invalid.
- If in every branch of proof a contradiction reached, F is valid.

Semantic Argument: Proof rules

$$\frac{I \models \neg F}{I \not\models F} \qquad \frac{I \not\models \neg F}{I \models F}$$

$$\frac{I \models F \wedge G}{\begin{array}{l} I \models F \\ I \models G \end{array}} \leftarrow \text{and} \qquad \frac{I \not\models F \wedge G}{I \not\models F \quad | \quad I \not\models G} \; \nwarrow \text{or}$$

$$\frac{I \models F \vee G}{I \models F \quad | \quad I \models G} \qquad \frac{I \not\models F \vee G}{\begin{array}{l} I \not\models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \rightarrow G}{I \not\models F \quad | \quad I \models G} \qquad \frac{I \not\models F \rightarrow G}{\begin{array}{l} I \models F \\ I \not\models G \end{array}}$$

$$\frac{I \models F \leftrightarrow G}{I \models F \wedge G \quad | \quad I \not\models F \vee G} \qquad \frac{I \not\models F \leftrightarrow G}{I \models F \wedge \neg G \quad | \quad I \models \neg F \wedge G}$$

$$\frac{\begin{array}{l} I \models F \\ I \not\models F \end{array}}{I \models \bot}$$

Example

Prove $\quad F : P \wedge Q \rightarrow P \vee \neg Q \quad$ is valid.

Let's assume that F is not valid and that I is a falsifying interpretation.

1.	$I \not\models P \wedge Q \rightarrow P \vee \neg Q$	assumption
2.	$I \models P \wedge Q$	1, Rule $\rightarrow$
3.	$I \not\models P \vee \neg Q$	1, Rule $\rightarrow$
4.	$I \models P$	2, Rule $\wedge$
5.	$I \not\models P$	3, Rule $\vee$
6.	$I \models \bot$	4 and 5 are contradictory

Thus F is valid.

Example 2

Prove $\quad F : (P \to Q) \land (Q \to R) \to (P \to R) \quad$ is valid.

Let's assume that F is not valid.

1.	$I \not\models F$	assumption
2.	$I \models (P \to Q) \land (Q \to R)$	1, Rule $\to$
3.	$I \not\models P \to R$	1, Rule $\to$
4.	$I \models P$	3, Rule $\to$
5.	$I \not\models R$	3, Rule $\to$
6.	$I \models P \to Q$	2, Rule $\land$
7.	$I \models Q \to R$	2, Rule $\land$

$8a.\quad I \not\models P$	$8b.\quad I \models Q \qquad 6 \to$	
$9a.\quad I \models \bot$	$9ba.\quad I \not\models Q$	$9bb.\quad I \models R$
	$10ba.\quad I \models \bot$	$10bb.\quad I \models \bot$

Our assumption is incorrect in all cases — F is valid.

Example 3

Is $\quad F : P \vee Q \rightarrow P \wedge Q \quad$ valid?

Let's assume that F is not valid.

$$\begin{array}{lll}
1. & I \not\models P \vee Q \rightarrow P \wedge Q & \text{assumption} \\
2. & I \models P \vee Q & 1 \text{ and } \rightarrow \\
3. & I \not\models P \wedge Q & 1 \text{ and } \rightarrow
\end{array}$$

$$\begin{array}{c|c}
\begin{array}{c}
4a. \quad I \models P \qquad 2 \text{ and } \vee \\
\begin{array}{l|l}
5aa. \; I \not\models P & 5ab. \; I \not\models Q \\
6aa. \; I \models \bot &
\end{array}
\end{array}
&
\begin{array}{c}
4b. \quad I \models Q \qquad 2 \text{ and } \vee \\
\begin{array}{l|l}
5ba. \; I \not\models P & 5bb. \; I \not\models Q \\
 & 6bb. \; I \models \bot
\end{array}
\end{array}
\end{array}$$

We cannot always derive a contradiction. F is not valid.

Falsifying interpretation:

$I_1 : \{P \mapsto \text{true}, \; Q \mapsto \text{false}\} \qquad I_2 : \{Q \mapsto \text{true}, \; P \mapsto \text{false}\}$

We have to derive a contradiction in all cases for F to be valid.

Normal Forms

Idea: Simplify decision procedure, by simplifying the formula first.
Convert it into a simpler normal form, e.g.:

- Negation Normal Form: No $\rightarrow$ and no $\leftrightarrow$; negation only before atoms.
- Conjunctive Normal Form: Negation normal form, where conjunction is outside, disjunction is inside.
- Disjunctive Normal Form: Negation normal form, where disjunction is outside, conjunction is inside.

The formula in normal form should be equivalent to the original input.

Equivalence

F_1 and F_2 are equivalent ($F_1 \Leftrightarrow F_2$)
iff for all interpretations I, $I \models F_1 \leftrightarrow F_2$

To prove $F_1 \Leftrightarrow F_2$ show $F_1 \leftrightarrow F_2$ is valid.

F_1 implies F_2 ($F_1 \Rightarrow F_2$)
iff for all interpretations I, $I \models F_1 \rightarrow F_2$

$F_1 \Leftrightarrow F_2$ and $F_1 \Rightarrow F_2$ are not formulae!

Equivalence is a Congruence relation

If $F_1 \Leftrightarrow F_1'$ and $F_2 \Leftrightarrow F_2'$, then

- $\neg F_1 \Leftrightarrow \neg F_1'$
- $F_1 \vee F_2 \Leftrightarrow F_1' \vee F_2'$
- $F_1 \wedge F_2 \Leftrightarrow F_1' \wedge F_2'$
- $F_1 \rightarrow F_2 \Leftrightarrow F_1' \rightarrow F_2'$
- $F_1 \leftrightarrow F_2 \Leftrightarrow F_1' \leftrightarrow F_2'$

- if we replace in a formula F a subformula F_1 by F_1' and obtain F', then $F \Leftrightarrow F'$.

Negation Normal Form (NNF)

Negations appear only in literals. (only $\neg, \wedge, \vee$)

To transform F to equivalent F' in NNF use recursively the following template equivalences (left-to-right):

$$\neg\neg F_1 \Leftrightarrow F_1 \qquad \neg\top \Leftrightarrow \bot \qquad \neg\bot \Leftrightarrow \top$$

$$\left.\begin{array}{l} \neg(F_1 \wedge F_2) \Leftrightarrow \neg F_1 \vee \neg F_2 \\ \neg(F_1 \vee F_2) \Leftrightarrow \neg F_1 \wedge \neg F_2 \end{array}\right\}\text{De Morgan's Law}$$

$$F_1 \rightarrow F_2 \Leftrightarrow \neg F_1 \vee F_2$$

$$F_1 \leftrightarrow F_2 \Leftrightarrow (F_1 \rightarrow F_2) \wedge (F_2 \rightarrow F_1)$$

Example: Negation Normal Form

Convert $F : (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \to R_2)$ into NNF

$$\begin{array}{rl} & (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \to R_2) \\ \Leftrightarrow & (Q_1 \lor R_1) \land (\neg Q_2 \to R_2) \\ \Leftrightarrow & (Q_1 \lor R_1) \land (\neg\neg Q_2 \lor R_2) \\ \Leftrightarrow & (Q_1 \lor R_1) \land (Q_2 \lor R_2) \end{array}$$

The last formula is equivalent to F and is in NNF.

Disjunctive Normal Form (DNF)

Disjunction of conjunctions of literals

$$\bigvee_i \bigwedge_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

To convert F into equivalent F' in DNF,
transform F into NNF and then
use the following template equivalences (left-to-right):

$$\left.\begin{array}{r} (F_1 \vee F_2) \wedge F_3 \Leftrightarrow (F_1 \wedge F_3) \vee (F_2 \wedge F_3) \\ F_1 \wedge (F_2 \vee F_3) \Leftrightarrow (F_1 \wedge F_2) \vee (F_1 \wedge F_3) \end{array}\right\} \mathit{dist}$$

Example

Convert $F : (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \rightarrow R_2)$ into DNF

$$
\begin{array}{lll}
& (Q_1 \lor \neg\neg R_1) \land (\neg Q_2 \rightarrow R_2) & \\
\Leftrightarrow & (Q_1 \lor R_1) \land (Q_2 \lor R_2) & \text{in NNF} \\
\Leftrightarrow & (Q_1 \land (Q_2 \lor R_2)) \lor (R_1 \land (Q_2 \lor R_2)) & \text{dist} \\
\Leftrightarrow & (Q_1 \land Q_2) \lor (Q_1 \land R_2) \lor (R_1 \land Q_2) \lor (R_1 \land R_2) & \text{dist}
\end{array}
$$

The last formula is equivalent to F and is in DNF. Note that formulas can grow exponentially.

Conjunctive Normal Form (CNF)

UNI FREIBURG

Conjunction of disjunctions of literals

$$\bigwedge_i \bigvee_j \ell_{i,j} \quad \text{for literals } \ell_{i,j}$$

To convert F into equivalent F' in CNF,
transform F into NNF and then
use the following template equivalences (left-to-right):

$$(F_1 \wedge F_2) \vee F_3 \Leftrightarrow (F_1 \vee F_3) \wedge (F_2 \vee F_3)$$
$$F_1 \vee (F_2 \wedge F_3) \Leftrightarrow (F_1 \vee F_2) \wedge (F_1 \vee F_3)$$

A disjunction of literals $P_1 \vee P_2 \vee \neg P_3$ is called a clause.
For brevity we write it as set: $\{P_1, P_2, \overline{P_3}\}$.
A formula in CNF is a set of clauses (a set of sets of literals).

Equisatisfiability

Definition (Equisatisfiability)

F and F' are equisatisfiable, iff

$$F \text{ is satisfiable if and only if } F' \text{ is satisfiable}$$

Every formula is equisatifiable to either $\top$ or $\bot$.

There is a efficient conversion of F to F' where

- F' is in CNF and
- F and F' are equisatisfiable

Note: efficient means polynomial in the size of F.

Conversion to CNF

Basic Idea:

- Introduce a new variable P_G for every subformula G; unless G is already an atom.
- For each subformula $G : G_1 \circ G_2$ produce a small formula $P_G \leftrightarrow P_{G_1} \circ P_{G_2}$.
- encode each of these (small) formulae separately to CNF.

The formula

$$P_F \wedge \bigwedge_G CNF(P_G \leftrightarrow P_{G_1} \circ P_{G_2})$$

is equisatisfiable to F.

The number of subformulae is linear in the size of F.

The time to convert one small formula is constant!

Example: CNF

Convert $F : P \vee Q \rightarrow P \wedge \neg R$ to CNF.
Introduce new variables: P_F, $P_{P \vee Q}$, $P_{P \wedge \neg R}$, $P_{\neg R}$. Create new formulae and convert them to CNF separately:

- $P_F \leftrightarrow (P_{P \vee Q} \rightarrow P_{P \wedge \neg R})$ in CNF:

$$F_1 : \{\{\overline{P_F}, \overline{P_{P \vee Q}}, P_{P \wedge \neg R}\}, \{P_F, P_{P \vee Q}\}, \{P_F, \overline{P_{P \wedge \neg R}}\}\}$$

- $P_{P \vee Q} \leftrightarrow P \vee Q$ in CNF:

$$F_2 : \{\{\overline{P_{P \vee Q}}, P \vee Q\}, \{P_{P \vee Q}, \overline{P}\}, \{P_{P \vee Q}, \overline{Q}\}\}$$

- $P_{P \wedge \neg R} \leftrightarrow P \wedge P_{\neg R}$ in CNF:

$$F_3 : \{\{\overline{P_{P \wedge \neg R}} \vee P\}, \{\overline{P_{P \wedge \neg R}}, P_{\neg R}\}, \{P_{P \wedge \neg R}, \overline{P}, \overline{P_{\neg R}}\}\}$$

- $P_{\neg R} \leftrightarrow \neg R$ in CNF: $F_4 : \{\{\overline{P_{\neg R}}, \overline{R}\}, \{P_{\neg R}, R\}\}$

$\{\{P_F\}\} \cup F_1 \cup F_2 \cup F_3 \cup F_4$ is in CNF and equisatisfiable to F.

Decides the satisfiability of PL formulae in CNF

Decision Procedure DPLL: Given F in CNF

```
let rec DPLL F =
  let F' = PROP F in
  let F'' = PLP F' in
  if F'' = ⊤ then true
  else if F'' = ⊥ then false
  else
     let P = CHOOSE vars(F'') in
     (DPLL F''{P ↦ ⊤}) ∨ (DPLL F''{P ↦ ⊥})
```

Unit Propagagion

Unit Propagation (PROP)

If a clause contains one literal ℓ,

- Set ℓ to $\top$.
- Remove all clauses containing ℓ.
- Remove $\neg\ell$ in all clauses.

Based on resolution

$$\frac{\ell \qquad \neg\ell \lor C}{C} \leftarrow \text{clause}$$

Pure Literal Propagagion

Pure Literal Propagation (PLP)

If P occurs only positive (without negation), set it to $\top$.
If P occurs only negative set it to $\bot$.

Example

$$F : (\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg Q \lor \neg R)$$

Branching on Q

$$F\{Q \mapsto \top\} : (R) \land (\neg R) \land (P \lor \neg R)$$

By unit resolution

$$\frac{R \qquad (\neg R)}{\bot}$$

$F\{Q \mapsto \top\} = \bot \Rightarrow$ false

On the other branch

$F\{Q \mapsto \bot\} : (\neg P \lor R)$

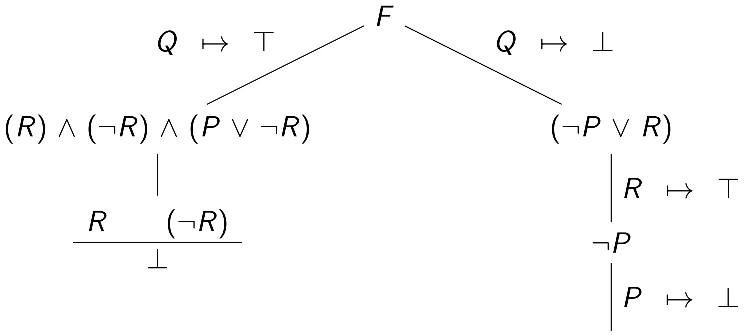
$F\{Q \mapsto \bot, R \mapsto \top, P \mapsto \bot\} = \top \Rightarrow$ true

F is satisfiable with satisfying interpretation

$$I : \{P \mapsto \text{false}, Q \mapsto \text{false}, R \mapsto \text{true}\}$$

Example

$$F : (\neg P \lor Q \lor R) \land (\neg Q \lor R) \land (\neg Q \lor \neg R) \land (P \lor \neg Q \lor \neg R)$$



$$I : \{P \mapsto \text{false},\ Q \mapsto \text{false},\ R \mapsto \text{true}\}$$

A island is inhabited only by knights and knaves. Knights always tell the truth, and knaves always lie. You meet four inhabitants: Alice, Bob, Charles and Doris.

- Alice says that Doris is a knave.
- Bob tells you that Alice is a knave.
- Charles claims that Alice is a knave.
- Doris tells you, ‘Of Charles and Bob, exactly one is a knight.’

Knight and Knaves

Let A denote that Alice is a Knight, etc. Then:

- $A \leftrightarrow \neg D$
- $B \leftrightarrow \neg A$
- $C \leftrightarrow \neg A$
- $D \leftrightarrow \neg(C \leftrightarrow B)$

In CNF:

- $\{\overline{A}, \overline{D}\}, \{A, D\}$
- $\{\overline{B}, \overline{A}\}, \{B, A\}$
- $\{\overline{C}, \overline{A}\}, \{C, A\}$
- $\{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}$

Solving Knights and Knaves

$$F : \{\{\overline{A}, \overline{D}\}, \{A, D\}, \{\overline{B}, \overline{A}\}, \{B, A\}, \{\overline{C}, \overline{A}\}, \{C, A\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$$

PROP and PLP are not applicable. Decide on A:

$F\{A \mapsto \bot\} : \{\{D\}, \{B\}, \{C\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$

By PROP we get:

$$F\{A \mapsto \bot, D \mapsto \top, B \mapsto \top, C \mapsto \top\} : \bot$$

Unsatisfiable! Now set A to $\top$:

$F\{A \mapsto \top\} : \{\{\overline{D}\}, \{\overline{B}\}, \{\overline{C}\}, \{\overline{D}, \overline{C}, \overline{B}\}, \{\overline{D}, C, B\}, \{D, \overline{C}, B\}, \{D, C, \overline{B}\}\}$

By PROP we get:

$$F\{A \mapsto \top, D \mapsto \bot, B \mapsto \bot, C \mapsto \bot\} : \top$$

Satisfying assignment!

Learning is Useful

Consider the following problem:

$$\{\{A_1, B_1\}, \{\overline{P_0}, \overline{A_1}, P_1\}, \{\overline{P_0}, \overline{B_1}, P_1\}, \{A_2, B_2\}, \{\overline{P_1}, \overline{A_2}, P_2\}, \{\overline{P_1}, \overline{B_2}, P_2\},$$
$$\dots, \{A_n, B_n\}, \{\overline{P_{n-1}}, \overline{A_n}, P_n\}, \{\overline{P_{n-1}}, \overline{B_n}, P_n\}, \{P_0\}, \{\overline{P_n}\}\}$$

For some literal orderings, we need exponentially many steps.
Note, that

$$\{\{A_i, B_i\}, \{\overline{P_{i-1}}, \overline{A_i}, P_i\}, \{\overline{P_{i-1}}, \overline{B_i}, P_i\}\} \Rightarrow \{\{\overline{P_{i-1}}, P_i\}\}$$

If we learn the right clauses, unit propagation will immediately give unsatisfiable.

Partial Assignments and Unit/Conflict Clauses

Do not change the clause set, but only assign literals (as global variables).
When you assign true to a literal ℓ,also assign false to $\bar{\ell}$.
For a partial assignment

- A clause is true if one of its literals is assigned true.
- A clause is a conflict clause if all its literals are assigned false.
- A clause is a unit clause if all but one literals are assigned false and the last literal is unassigned.

If the assignment of a literal from a conflict clause is removed we get a unit clause.
Explain unsatisfiability of partial assignment by conflict clause and learn it!

Conflict Driven Clause Learning (CDCL)

Idea: Explain unsatisfiability of partial assignment by conflict clause and learn it!

- If a conflict is found we return the conflict clause.
- If variable in conflict were derived by unit propagation use resolution rule to generate a new conflict clause.
- If variable in conflict was derived by decision, use learned conflict as unit clause

DPLL with CDCL

The functions DPLL and PROP return a conflict clause or satisfiable.

```
let rec DPLL =
  let PROP U =
    ...
  if conflictclauses ≠ ∅
    CHOOSE conflictclauses
  else if unitclauses ≠ ∅
    PROP (CHOOSE unitclauses)
  else if coreclauses ≠ ∅
     let ℓ = CHOOSE (⋃coreclauses) ∩ unassigned in
     val[ℓ] := ⊤
     let C = DPLL in
     if (C = satisfiable) satisfiable
     else
        val[ℓ] := undef
        if (ℓ̄ ∉ C) C
        else LEARN C; PROP C
  else satisfiable
```

Unit propagation

The function PROP takes a unit clause and does unit propagation. It calls DPLL recursively and returns a conflict clause or satisfiable. recursively:

```
let PROP U =
  let ℓ = CHOOSE U ∩ unassigned in
  val[ℓ] := ⊤
  let C = DPLL in
  if (C = satisfiable)
    satisfiable
  else
    val[ℓ] := undef
    if (ℓ̄ ∉ C) C
    else U \ {ℓ} ∪ C \ {ℓ̄}
```

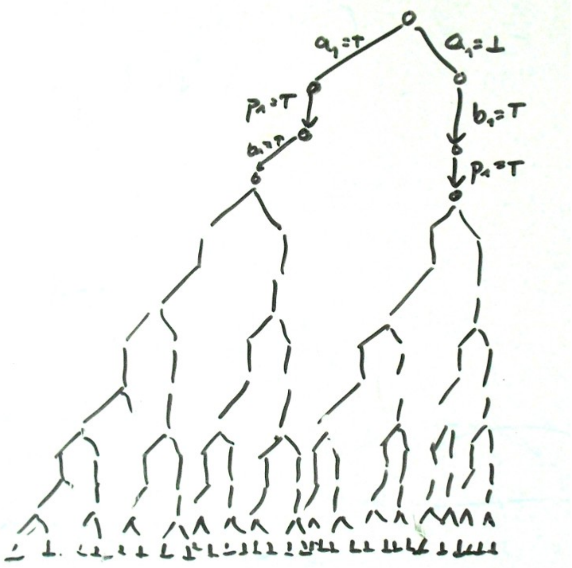
The last line does resolution:

$$\frac{\ell \lor C_1 \qquad \neg\ell \lor C_2}{C_1 \lor C_2}$$

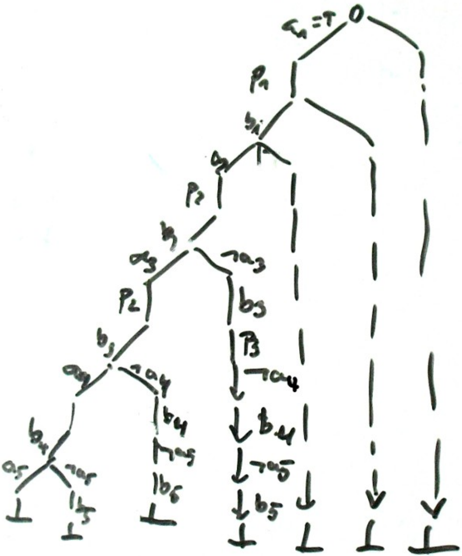
$$\{\{A_1, B_1\}, \{\overline{P_0}, \overline{A_1}, P_1\}, \{\overline{P_0}, \overline{B_1}, P_1\}, \{A_2, B_2\}, \{\overline{P_1}, \overline{A_2}, P_2\}, \{\overline{P_1}, \overline{B_2}, P_2\}, \ldots, \{A_n, B_n\}, \{\overline{P_{n-1}}, \overline{A_n}, P_n\}, \{\overline{P_{n-1}}, \overline{B_n}, P_n\}, \{P_0\}, \{\overline{P_n}\}\}$$

- Unit propagation (PROP) sets P_0 and $\overline{P_n}$ to true.
- Decide, e.g. A_1, PROP sets $\overline{P_1}$
- Continue until A_{n-1}, PROP sets $\overline{P_{n-1}}, \overline{A_n}$ and $\overline{B_n}$
- Conflict clause computed: $\{\overline{A_{n-1}}, \overline{P_{n-2}}, P_n\}$.
- Conflict clause does not depend on $A_1, \ldots, A_{n-2}$ and can be used again.

DPLL (without Learning)



DPLL with CDCL



Some Notes about DPLL with Learning

- Pure Literal Propagation is unnecessary:
  A pure literal is always chosen right and never causes a conflict.
- Modern SAT-solvers use this procedure but differ in
  - heuristics to choose literals/clauses.
  - efficient data structures to find unit clauses.
  - more conflict resolution to minimize learned clauses.
  - restarts (without forgetting learned clauses).
- Even with the optimal heuristics DPLL is still exponential:
  The Pidgeon-Hole problem requires exponential resolution proofs.

Summary

- Syntax and Semantics of Propositional Logic
- Methods to decide satisfiability/validity of formulae:
  - Truth table
  - Semantic Tableaux
  - DPLL
- Run-time of all algorithm is worst-case exponential in length of formula.
- Deciding satisfiability is NP-complete.