03—Propositional Logic III

CS 5209: Foundation in Logic and AI

Martin Henz and Aqinas Hobor

January 28, 2010

Generated on Monday 1st February, 2010, 16:37

Conjunctive Normal Form

Definition

A literal L is either an atom p or the negation of an atom $\neg p$. A formula C is in *conjunctive normal form* (CNF) if it is a conjunction of clauses, where each clause is a disjunction of literals:

$$\begin{array}{rcl} L & ::= & p | \neg p \\ D & ::= & L | L \vee D \\ C & ::= & D | D \wedge C \end{array}$$

Examples

$(\neg p \vee q \vee r) \wedge (\neg q \vee r) \wedge (\neg r)$ is in CNF.
$(\neg p \vee q \vee r) \wedge ((p \wedge \neg q) \vee r) \wedge (\neg r)$ is not in CNF.
$(\neg p \vee q \vee r) \wedge \neg(\neg q \vee r) \wedge (\neg r)$ is not in CNF.

Usefulness of CNF

Lemma

A disjunction of literals $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$.

How to disprove

$$\models (\neg q \vee p \vee r) \wedge (\neg p \vee r) \wedge q$$

Disprove any of:

$$\models (\neg q \vee p \vee r) \qquad \models (\neg p \vee r) \qquad \models q$$

Usefulness of CNF

Lemma

A disjunction of literals $L_1 \vee L_2 \vee \cdots \vee L_m$ is valid iff there are $1 \leq i, j \leq m$ such that L_i is $\neg L_j$.

How to prove

$$\models (\neg q \vee p \vee q) \wedge (p \vee r \neg p) \wedge (r \vee \neg r)$$

Prove all of:

$$\models (\neg q \vee p \vee q) \qquad \models (p \vee r \neg p) \qquad \models (r \vee \neg r)$$

Usefulness of CNF

Proposition

Let ϕ be a formula of propositional logic. Then ϕ is satisfiable iff $\neg\phi$ is not valid.

Satisfiability test

We can test satisfiability of ϕ by transforming $\neg\phi$ into CNF, and show that some clause is not valid.

Transformation to CNF

Theorem

Every formula in the propositional calculus can be transformed into an equivalent formula in CNF.

Algorithm for CNF Transformation

1. Eliminate implication using:
   $A \rightarrow B \equiv \neg A \vee B$
2. Push all negations inward using De Morgan's laws:
   $\neg(A \wedge B) \equiv (\neg A \vee \neg B)$
   $\neg(A \vee B) \equiv (\neg A \wedge \neg B)$
3. Eliminate double negations using the equivalence $\neg\neg A \equiv A$
4. The formula now consists of disjunctions and conjunctions of literals. Use the distributive laws
   $A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$
   $(A \wedge B) \vee C \equiv (A \vee C) \wedge (B \vee C)$
   to eliminate conjunctions within disjunctions.

Example

$$
\begin{aligned}
(\neg p \rightarrow \neg q) \rightarrow (p \rightarrow q) &\equiv \neg(\neg\neg p \vee \neg q) \vee (\neg p \vee q) \\
&\equiv (\neg\neg\neg p \wedge q) \vee (\neg p \vee q) \\
&\equiv (\neg p \wedge q) \vee (\neg p \vee q) \\
&\equiv (\neg p \vee \neg p \vee q) \wedge (q \vee \neg p \vee q)
\end{aligned}
$$

WalkSAT: An Incomplete Solver

Idea: Start with a random truth assignment, and then iteratively improve the assignment until model is found

Details: In each step, choose an unsatisfied clause (clause selection), and “flip” one of its variables (variable selection).

WalkSAT: Details

Termination criterion: No unsatisfied clauses are left.

Clause selection: Choose a random unsatisfied clause.

Variable selection:

- If there are variables that when flipped make no currently satisfied clause unsatisfied, flip one which makes the most unsatisfied clauses satisfied.
- Otherwise, make a choice with a certain probability between:
  - picking a random variable, and
  - picking a variable that when flipped minimizes the number of unsatisfied clauses.

DPLL: Idea

- Simplify formula based on pure literal elimination and unit propagation
- If not done, pick an atom *p* and split: $\phi \wedge p$ or $\phi \wedge \neg p$

A Linear Solver: Idea

- Transform formula to tree of conjunctions and negations.
- Transform tree into graph.
- Mark the top of the tree as T.
- Propagate constraints using obvious rules.
- If all leaves are marked, check that corresponding assignment makes the formula true.

Transformation

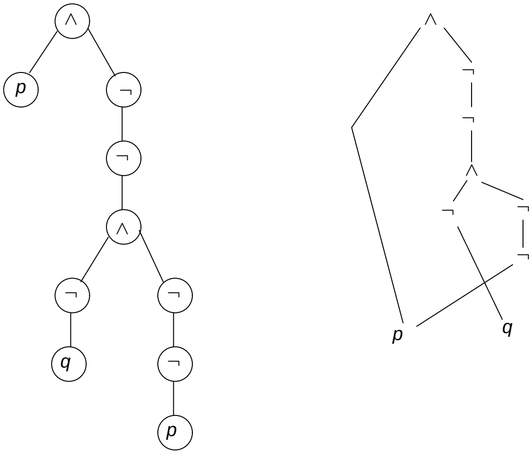
$$
\begin{array}{rcl}
T(p) & = & p \\
T(\phi_1 \wedge \phi_2) & = & T(\phi_1) \wedge T(\phi_2) \\
T(\neg\phi) & = & \neg\phi(T) \\
T(\phi_1 \rightarrow \phi_2) & = & \neg(T(\phi_1) \wedge \neg T(\phi_2)) \\
T(\phi_1 \vee \phi_2) & = & \neg(\neg T(\phi_1) \wedge \neg T(\phi_2))
\end{array}
$$

Example

$$\phi = p \land \neg(q \lor \neg p)$$

$$T(\phi) = p \land \neg\neg(\neg q \land \neg\neg p)$$

Binary Decision Tree: Example



Problem

What happens to formulas of the kind $\neg(\phi_1 \wedge \phi_2)$?

A Cubic Solver: Idea

Improve the linear solver as follows:

- Run linear solver
- For every node *n* that is still unmarked:
  - Mark *n* with T and run linear solver, possibly resulting in temporary marks.
  - Mark *n* with F and run linear solver, possibly resulting in temporary marks.
  - Combine temporary marks, resulting in possibly new permanent marks

The ACC 1997/98 Problem

- “ACC” stands for “Atlantic Coast Conference”, an American college basketball organization
- 9 teams participate in tournament
- dense double round robin: there are $2 * 9$ dates
- at each date, each team plays either home, away or has a “bye”
- Each team must play each other team once at home and once away.
- there should be at least 7 dates distance between first leg and return match.
- To achieve this, we assume a fixed mirroring between dates: (1,8), (2,9), (3,12), (4,13), (5,14), (6,15) (7,16), (10,17), (11,18)

The ACC 1997/98 Problem (contd)

- No team can play away on both last dates
- No team may have more than two away matches in a row.
- No team may have more than two home matches in a row.
- No team may have more than three away matches or byes in a row.
- No team may have more than four home matches or byes in a row.

The ACC 1997/98 Problem (contd)

- Of the weekends, each team plays four at home, four away, and one bye.
- Each team must have home matches or byes at least on two of the first five weekends.
- Every team except FSU has a traditional rival. The rival pairs are Clem-GT, Duke-UNC, UMD-UVA and NCSt-Wake. In the last date, every team except FSU plays against its rival, unless it plays against FSU or has a bye.

The ACC 1997/98 Problem (contd)

- The following pairings must occur at least once in dates 11 to 18: Duke-GT, Duke-Wake, GT-UNC, UNC-Wake.
- No team plays in two consecutive dates away against Duke and UNC. No team plays in three consecutive dates against Duke UNC and Wake.
- UNC plays Duke in last date and date 11.
- UNC plays Clem in the second date.
- Duke has bye in the first date 16.

The ACC 1997/98 Problem (contd)

- Wake does not play home in date 17.
- Wake has a bye in the first date.
- Clem, Duke, UMD and Wake do not play away in the last date.
- Clem, FSU, GT and Wake do not play away in the fist date.
- Neither FSU nor NCSt have a bye in the last date.
- UNC does not have a bye in the first date.

Background

- Trick and Nemhauser work on the problem from 1995 onwards
- Trick and Nemhauser publish the problem and their approach in “Scheduling a Major Basketball Conference”, Operations Research, 46(1), 1998
- From then onwards, Henz, Walser and Zhang use different techniques to solve the problem

General Approach

- Three phases:
  1. Generate all possible patterns such as
     “A H B A H H A H A A H B H A A H H A”
  2. Generate all feasible 9-element pattern sets that can be used to construct a schedule
  3. Generate schedules from pattern sets
- Output: all feasible solutions, from which the organizers can choose the most suitable one

Solution Techniques

- Nemhauser and Trick use integer programming for all three steps, leading to a “turn-around time” of 24 hours
- Henz uses constraint programming, turn-around time of less than 1 minute, publishes his approach in “Scheduling a Major Basketball Conference—Revisited”, Operations Research, 49(1), 2001
- Zhang Hantao uses SAT solving, turn-around time of 2 seconds, see “Generating College Conference Basketball Schedules using a SAT Solver”
- Different approach: In 1998, J.P. Walser described a local-search based method for finding some (not all) solutions, without using 3 phases

How to Encode ACC as a SAT Formula

- Consider Phase 3: Generation of schedule, assigning teams to opponents at every day of the tournament
- For teams x, y, day z, introduce atom $p_{x,y,z} = T$ iff team x plays a home game against team y in day z.
- Example of encoding constraints: "Each team must play each other team once at home and once away."
- For every pair of distinct teams s and t, we have:

$$
\begin{gathered}
(p_{s,t,1} \wedge \neg p_{s,t,2} \wedge \cdots \wedge \neg p_{s,t,18}) \vee \\
(\neg p_{s,t,1} \wedge p_{s,t,2} \wedge \neg p_{s,t,3} \wedge \cdots \wedge \neg p_{s,t,18}) \vee \\
\vdots \\
(\neg p_{s,t,1} \cdots \wedge \neg p_{s,t,17} \wedge p_{s,t,18})
\end{gathered}
$$

- Convert formula into CNF and use a complete SAT solver

Some Statistics

- Zhang Hantao used the DPLL-based SAT solver SATO
- Phase 1: $18 \cdot 3 = 54$ propositional atoms, 1499 clauses, taking 0.01 seconds, resulting in 38 patterns
- Phase 2: $38 \cdot 9 \cdot 3 = 1026$ propositional atoms, 569300 clauses, taking 0.60 seconds, resulting in 17 pattern sets
- Phase 3: $9 \cdot 9 + 9 \cdot 8 \cdot 18 = 1377$ propositional atoms, hundreds of thousands of clauses, taking less than 2 seconds, resulting in 179 solutions

Conclusion

- For many discrete constraint satisfaction problems such as the ACC 1997/98 problem, an encoding in SAT and use of a state-of-the-art SAT solver provides an attractive solving technique.
- The approach takes advantage of the effort that the designers of SAT solvers such as SATO spent in order to optimize the solver.
- This works well, because the solver is independent of the application domain; it can be used without modification across application domains.