CS208 (Semester 1) Week 3 : Logical Modelling II

Dr. Robert Atkey

Computer & Information Sciences

Logical Modelling II, Part 1

Conversion to CNF

Conjunctive Normal Form (CNF)

$$
\begin{array}{ll}
 & (\neg a \lor \neg b \lor \neg c) \\
\land & (\neg b \lor \neg c \lor \neg d) \\
\land & (\neg a \lor \neg b \lor c) \\
\land & b
\end{array}
$$

1. Entire formula is a conjunction $C_1 \land C_2 \land \cdots \land C_n$
2. where each *clause* $C_i = L_{i,1} \lor L_{i,2} \lor \cdots \lor L_{i,k}$
3. where each *literal* $L_{i,j} = x_{i,j}$ or $L_{i,j} = \neg x_{i,j}$

Disjunctive Normal Form (DNF)

Disjunctive Normal Form (DNF) is similar, but swaps $\land$ and $\lor$.

$$
\begin{aligned}
& (\neg a \land \neg b \land \neg c) \\
\lor\ & (\neg b \land \neg c \land \neg d) \\
\lor\ & (\neg a \land \neg b \land c) \\
\lor\ & b
\end{aligned}
$$

1. Entire formula is a *dis*junction $D_1 \lor D_2 \lor \cdots \lor D_n$
2. where each *disjunct* $D_i = L_{i,1} \land L_{i,2} \land \cdots \land L_{i,k}$
3. where each *literal* $L_{i,j} = x_{i,j}$ or $L_{i,j} = \neg x_{i,j}$

Normal Forms and Satisfiability

CNF

Each clause is a *constraint* and all constraints must be satisfied.

DNF

At least one of the disjuncts must be satisfied.

Exercise (after all the videos): How would you write a SAT Solver for formulas in DNF? Why don't we do this instead of CNF?

Conversion to CNF

Not every formula is in CNF, e.g.,

$$(A \land B) \to (B \land A)$$

What if we want to use a SAT solver to determine satisfiability?

Two ways to convert a formula to CNF that is "the same":

- "Multiplying out"
- Tseytin transformation

First we need to define what we mean by "the same".

Equivalent Formulas

Define two formulas P and Q to be *equivalent*, written

$$P \equiv Q$$

exactly when, for all valuations v,

$$[\![P]\!]v = [\![Q]\!]v$$

Equivalent to both $P \models Q$ and $Q \models P$ being valid

Simplifying Implication

$$A \to B \equiv \neg A \vee B$$

valuation			P	Q
A	B	$\neg A$	$A \to B$	$\neg A \vee B$
F	F	T	T	T
F	T	T	T	T
T	F	F	F	F
T	T	F	T	T

Double Negation

Negating twice is the same as doing nothing:

$$A \equiv \neg\neg A$$

valuation		P	Q
A	$\neg A$	A	$\neg\neg A$
F	T	F	F
T	F	T	T

de Morgan's laws

Negation swaps $\wedge$ and $\vee$:

$$\neg(A \wedge B) \equiv \neg A \vee \neg B$$

valuation					P	Q
A	B	$\neg A$	$\neg B$	$A \wedge B$	$\neg(A \wedge B)$	$\neg A \vee \neg B$
F	F	T	T	F	T	T
F	T	T	F	F	T	T
T	F	F	T	F	T	T
T	T	F	F	T	F	F

Similar for $\neg(A \vee B) \equiv \neg A \wedge \neg B$

Negation Normal Form (NNF)

Using the equivalences:

$$
\begin{aligned}
A \to B &\equiv \neg A \vee B \\
A &\equiv \neg\neg A \\
\neg(A \wedge B) &\equiv \neg A \vee \neg B \\
\neg(A \vee B) &\equiv \neg A \wedge \neg B
\end{aligned}
$$

We can *rewrite* any formula into an equivalent one with

1. No implications ($\to$s)
2. All negation signs on the atomic propositions

Example

$$
\begin{array}{lll}
 & (a \wedge (a \rightarrow b)) \rightarrow c & \\
\equiv & \neg(a \wedge (a \rightarrow b)) \vee c & \textit{converted } \rightarrow \\
\equiv & \neg(a \wedge (\neg a \vee b)) \vee c & \textit{converted } \rightarrow \\
\equiv & \neg a \vee \neg(\neg a \vee b) \vee c & \textit{converted } \wedge \textit{ to } \vee \\
\equiv & \neg a \vee (\neg\neg a \wedge \neg b) \vee c & \textit{converted } \vee \textit{ to } \wedge \\
\equiv & \neg a \vee (a \wedge \neg b) \vee c & \textit{converted double negation}
\end{array}
$$

Now in NNF, but not CNF.

"Push" $\vee$s into $\wedge$s

$$A \vee (B \wedge C) \equiv (A \vee B) \wedge (A \vee C)$$

valuation						P	Q
A	B	C	$B \wedge C$	$A \vee B$	$A \vee C$	$A \vee (B \wedge C)$	$(A \vee B) \wedge (A \vee C)$
F	F	F	F	F	F	F	F
F	F	T	F	F	T	F	F
F	T	F	F	T	F	F	F
F	T	T	T	T	T	T	T
T	F	F	F	T	T	T	T
T	F	T	F	T	T	T	T
T	T	F	F	T	T	T	T
T	T	T	T	T	T	T	T

Conversion to CNF

$$
\begin{array}{ll}
 & \neg a \lor (a \land \neg b) \lor c \\
\equiv & \hfill \textit{multiply out} \\
 & \neg a \lor ((a \lor c) \land (\neg b \lor c)) \\
\equiv & \hfill \textit{multiply out} \\
 & (\neg a \lor a \lor c) \land (\neg a \lor \neg b \lor c)
\end{array}
$$

Now in CNF.

(Can further simplify to: $(\neg a \lor \neg b \lor c)$)

Exponential Blowup

University of Strathclyde Science

If we convert $(a \wedge b \wedge c) \vee (d \wedge e \wedge f) \vee (g \wedge h \wedge i)$ to CNF, we get:

$$\begin{array}{l}
(a \vee d \vee g) \wedge (a \vee d \vee h) \wedge (a \vee d \vee i) \wedge (a \vee e \vee g) \wedge (a \vee e \vee h) \wedge \\
(a \vee e \vee i) \wedge (a \vee f \vee g) \wedge (a \vee f \vee h) \wedge (a \vee f \vee i) \wedge (b \vee d \vee g) \wedge \\
(b \vee d \vee h) \wedge (b \vee d \vee i) \wedge (b \vee e \vee g) \wedge (b \vee e \vee h) \wedge (b \vee e \vee i) \wedge \\
(b \vee f \vee g) \wedge (b \vee f \vee h) \wedge (b \vee f \vee i) \wedge (c \vee d \vee g) \wedge (c \vee d \vee h) \wedge \\
(c \vee d \vee i) \wedge (c \vee e \vee g) \wedge (c \vee e \vee h) \wedge (c \vee e \vee i) \wedge (c \vee f \vee g) \wedge \\
(c \vee f \vee h) \wedge (c \vee f \vee i)
\end{array}$$

which has 27 clauses.

Summary

- SAT Solvers take their input in CNF
- Some problems are naturally in CNF
- Conversion by "multiplying out" can generate huge formulas
- We need something better

Logical Modelling II, Part 2

Tseytin Transformation

Tseytin Transformation

The Tseytin transformation converts a formula into CNF with at most 3 times as many clauses as connectives in the original formula (versus potentially exponential for multiplying out the brackets).

1. Convert the formula into equations

   One connective $\rightsquigarrow$ one equation

2. Convert each equation into clauses

   One equation $\rightsquigarrow$ 2-3 clauses

Result is not equivalent, but *equisatisfiable*.

1. Name subformulas

Take the formula and name all the non-atomic subformulas.

Example:

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

becomes:

$$\begin{aligned} x_1 &= x_2 \vee c \\ x_2 &= \neg x_3 \\ x_3 &= a \wedge x_4 \\ x_4 &= x_5 \vee b \\ x_5 &= \neg a \end{aligned}$$

2. Converting Equations to Clauses

Given an equation like $x = y \wedge z$, we want some clauses that are true for every valuation that satisfies the equation.

2. Converting Equations to Clauses

University of Strathclyde Science

Given an equation like $x = y \wedge z$, we want some clauses that are true for every valuation that satisfies the equation.

Derive by conversion to CNF:

$$\begin{aligned} & x = y \wedge z \\ \equiv\ & (x \to (y \wedge z)) \wedge ((y \wedge z) \to x) \\ \equiv\ & (\neg x \vee (y \wedge z)) \wedge (\neg(y \wedge z) \vee x) \\ \equiv\ & (\neg x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z \vee x) \end{aligned}$$

2. Equations to Clauses

Take each equation $x = y \mathbin{\Box} z$ and turn it into clauses:

1. If $x = y \wedge z$, add

$$(\neg x \vee y) \wedge (\neg x \vee z) \wedge (\neg y \vee \neg z \vee x)$$

2. If $x = y \vee z$, add

$$(y \vee z \vee \neg x) \wedge (\neg y \vee x) \wedge (\neg z \vee x)$$

3. If $x = \neg y$, add

$$(\neg y \vee \neg x) \wedge (y \vee x)$$

3. Assert the top level variable

If x is the name of the whole formula, add a clause with just x:

$$
\begin{array}{rl}
 & \mathit{equation\ 1} \\
\bigwedge & \mathit{equation\ 2} \\
\bigwedge & \ldots \\
\bigwedge & x
\end{array}
$$

This asserts that our original formula must be true.

Example: $\neg(A \wedge B) \vee (B \wedge A)$

1. Name the subformulas:

$$
\begin{array}{rclcrcl}
x_1 & = & x_2 \vee x_4 & \qquad & x_2 & = & \neg x_3 \\
x_3 & = & A \wedge B & \qquad & x_4 & = & B \wedge A
\end{array}
$$

Example: $\neg(A \wedge B) \vee (B \wedge A)$

1. Name the subformulas:

$$\begin{array}{rclcrcl} x_1 &=& x_2 \vee x_4 & & x_2 &=& \neg x_3 \\ x_3 &=& A \wedge B & & x_4 &=& B \wedge A \end{array}$$

2+3. Generate clauses: (One line per equation)

$$\begin{array}{ll} & (x_2 \vee x_4 \vee \neg x_1) \wedge (\neg x_2 \vee x_1) \wedge (\neg x_4 \vee x_1) \\ \wedge & (\neg x_3 \vee \neg x_2) \wedge (x_3 \vee x_2) \\ \wedge & (\neg A \vee \neg B \vee x_3) \wedge (A \vee \neg x_3) \wedge (B \vee \neg x_3) \\ \wedge & (\neg B \vee \neg A \vee x_4) \wedge (B \vee \neg x_4) \wedge (A \vee \neg x_4) \\ \wedge & x_1 \end{array}$$

Efficiency

In small examples, we get many clauses.

But we *always* get $\leq 3n$ clauses, where n number of connectives.

Multiplying out can result in exponential number of clauses.

Can also optimise (see the tutorial questions).

Not Equivalent!

The formulas generated by the Tseytin transformation are **not** equivalent to the original, because they have extra atomic propositions.

Example

If the original formula is

$$\neg A$$

the Tseytin transformed version is: (assuming we don't optimise)

$$(\neg A \vee \neg x) \wedge (A \vee x) \wedge x$$

Then $\{A : \mathsf{F}, x : \mathsf{F}\}$ satisfies the original, but not the transformed formula.

Equisatisfiable

If we write $\mathsf{Tseytin}(P)$ for the Tseytin translation of P, then:

1. If there exists a valuation v_1 such that $[\![P]\!]v_1 = \mathsf{T}$, then there exists a valuation v_2 such that $[\![\mathsf{Tseytin}(P)]\!]v_2 = \mathsf{T}$;
2. If there exists a valuation v such that $[\![\mathsf{Tseytin}(P)]\!]v = \mathsf{T}$, then the valuation $v' = v$ without the additional x_is makes $[\![P]\!]v' = \mathsf{T}$.

This is called "equisatisfiability".

Example

$v = \{A : \mathsf{F}\}$ satisfies $\neg A$

The corresponding satisfying valuation for

$$(\neg A \vee \neg x) \wedge (A \vee x) \wedge x$$

is $\{A : \mathsf{F}, x : \mathsf{T}\}$.

A corresponding satisfying assignment always exists for the Tseytin transformation, because it is built from equations.

Summary

- Tseytin transformation converts formulas to CNF
- Generates $\leq 3n$ clauses, where n is the number of connectives
- Avoids exponential blowup
- Can be further optimised
- Result is *equisatisfiable*

Logical Modelling II, Part 3

Online Satisfiability Checker