Propositional Calculus

Dr. Neil T. Dantam

CSCI-498/598 RPM, Colorado School of Mines

Spring 2018

Calculus?

Definition: Calculus

A well defined method for mathematical reasoning employing axioms and rules of inference or transformation. A **formal system** or **rewrite system**.

Examples:

- Differential calculus
- Lambda (λ) calculus
- Propositional calculus (Boolean/Propositional logic)
- Predicate calculus (First-Order logic)

Etymology: from the Latin "calx" (limestone) + "-ulus" (dimutive). A pebble or stone used for counting.

Outline

Boolean Variables

(propositions)

Values: $\mathbb{B} \equiv \{0, 1\}$

true: $1, T, \top$

false: $0, F, \bot$

Variables: $p \in \mathbb{B}$

$p_1, \ldots, p_n \in \mathbb{B}^n$

Boolean Operators

Basic

Unary: $f : \mathbb{B} \mapsto \mathbb{B}$

Binary: $g : \mathbb{B} \times \mathbb{B} \mapsto \mathbb{B}$

Not

- $\neg 0 = 1$
- $\neg 1 = 0$

And

- $0 \wedge 0 = 0$
- $0 \wedge 1 = 0$
- $1 \wedge 0 = 0$
- $1 \wedge 1 = 1$

Or

- $0 \vee 0 = 0$
- $0 \vee 1 = 1$
- $1 \vee 0 = 1$
- $1 \vee 1 = 1$

Boolean Operators

Extended

Xor

$$(a \oplus b) \triangleq (a \lor b) \land \neg(a \land b)$$
$$\triangleq (a \land \neg b) \lor (\neg a \land b)$$

- $0 \oplus 0 = 0$
- $0 \oplus 1 = 1$
- $1 \oplus 0 = 1$
- $1 \oplus 1 = 0$

Implies

$$(a \implies b) \triangleq (\neg a \lor b)$$

- $(0 \implies 0) = 1$
- $(0 \implies 1) = 1$
- $(1 \implies 0) = 0$
- $(1 \implies 1) = 1$

Biconditional (iff)

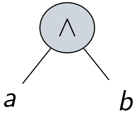
$$(a \iff b) \triangleq (a \implies b) \land (b \implies a)$$
$$\triangleq \neg(a \oplus b)$$
$$\triangleq (a \land b) \lor (\neg a \land \neg b)$$

- $(0 \iff 0) = 1$
- $(0 \iff 1) = 0$
- $(1 \iff 0) = 0$
- $(1 \iff 1) = 1$

Example: Boolean Formulae as S-expressions

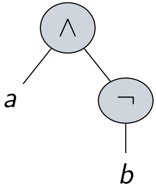
$$a \wedge b$$

```
(and a b)
```



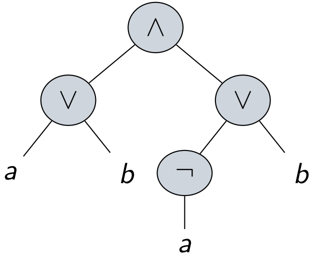
$$a \wedge \neg b$$

```
(and a (not b))
```



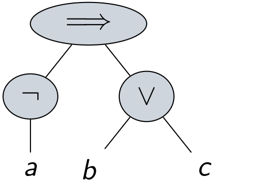
$$(a \vee b) \wedge (\neg a \vee b)$$

```
(and   (or a b)
       (or (not a) b))
```



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

```
(implies  (not a)
          (or b c))
```



Exercise: Boolean Formulae as S-expressions

$(a \land b) \implies c$	$\neg(a \land b) \lor c$	$\neg a \lor \neg b \lor c$

All these formulae are equivalent

N-ary Boolean Operators

AND

Infix		S-exp.
$\alpha \land \beta$	$=$	$(\texttt{AND}\ \alpha\ \beta)$
$\alpha \land \beta \land \gamma$	$=$	$(\texttt{AND}\ \alpha\ \beta\ \gamma)$
α	$=$	$(\texttt{AND}\ \alpha)$
?	$=$	$(\texttt{AND})$

OR

Infix		S-exp.
$\alpha \lor \beta$	$=$	$(\texttt{OR}\ \alpha\ \beta)$
$\alpha \lor \beta \lor \gamma$	$=$	$(\texttt{OR}\ \alpha\ \beta\ \gamma)$
α	$=$	$(\texttt{OR}\ \alpha)$
?	$=$	$(\texttt{OR})$

Identity Element

Arithmetic

Generally: $f(\alpha, \overbrace{\chi}^{\text{identity}}) = \alpha$

Multiplication

Infix:
- $a * \chi = a$
- $\chi = 1$

S-exp.:
- $(* \; a_0 \; \ldots \; a_n \; 1) = (* \; a_0 \; \ldots \; a_n)$

Addition

Infix:
- $a + \chi = a$
- $\chi = 0$

S-exp.:
- $(+ \; a_0 \; \ldots \; a_n \; 0) = (+ \; a_0 \; \ldots \; a_n)$

Identity Element

Boolean Algebra

$$\text{Generally: } f(\alpha, \overbrace{\chi}^{\text{identity}}) = \alpha$$

AND

Infix:
- $a \wedge \chi = a$
- $\chi = \top$

S-exp.:
- $(\texttt{AND}\ a_0\ \ldots\ a_n\ \top) = (\texttt{AND}\ a_0\ \ldots\ a_n)$

OR

Infix:
- $a \vee \chi = a$
- $\chi = \bot$

S-exp.:
- $(\texttt{OR}\ a_0\ \ldots\ a_n\ \bot) = (\texttt{OR}\ a_0\ \ldots\ a_n)$

Identity Element

Cancellation

AND

$$\chi = \top$$

$$
\begin{aligned}
& (\mathtt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \alpha_n) \\
\rightsquigarrow\ & \left(\mathtt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \cancelto{\top}{\alpha_n}\right)^{\top} \\
\rightsquigarrow\ & (\mathtt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}) \\
\rightsquigarrow\ & \left(\mathtt{AND}\ \cancelto{\top}{\alpha_0}\ \cancelto{\top}{\alpha_1}\ \ldots\ \cancelto{}{\alpha_{n-1}}\right)^{\top} \\
\rightsquigarrow\ & (\mathtt{AND}) \\
\rightsquigarrow\ & \top
\end{aligned}
$$

OR

$$\chi = \bot$$

$$
\begin{aligned}
& (\mathtt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \alpha_n) \\
\rightsquigarrow\ & \left(\mathtt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \cancelto{\bot}{\alpha_n}\right)^{\bot} \\
\rightsquigarrow\ & (\mathtt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}) \\
\rightsquigarrow\ & \left(\mathtt{OR}\ \cancelto{\bot}{\alpha_0}\ \cancelto{\bot}{\alpha_1}\ \ldots\ \cancelto{}{\alpha_{n-1}}\right)^{\bot} \\
\rightsquigarrow\ & (\mathtt{OR}) \\
\rightsquigarrow\ & \bot
\end{aligned}
$$

Remove canceled identity terms: base case is identity element

Describing Expressions

Literal:	A single variable or its negation
Conjunction:	An `AND` ($\wedge$) expression
Disjunction:	An `OR` ($\vee$) expression

Definitions

Literal

Definition (Literal)

A single variable or its negation.

Positive Literal: p

Negative Literal: $\neg p$

Examples

positive: p_i

negative: $\neg p_i$

Counter Examples

- $\cancel{p_i \wedge p_j}$
- $\cancel{p_i \vee p_j}$
- $\cancel{\neg(p_i \wedge p_j)}$

Definitions

Conjunction

Definition (Conjunction)

An n-ary AND. True when ALL of its arguments are true.

Examples

- $\left(p_i\right) = \left(p_i \wedge \bot\right)$
- $p_i \wedge p_j$
- $p_i \wedge p_j \wedge p_k$
- $p_i \wedge (p_j \vee p_k)$

Counter Examples

- $\cancel{p_i \vee p_j}$
- $\cancel{p_i \vee (p_j \wedge p_k)}$
- $\cancel{(p_i \wedge p_j) \vee p_k}$

Definitions

Disjunction

Definition (Disjunction)

An n-ary OR. True when ANY of its arguments are true.

Examples

- $\left(p_i\right) = \left(p_i \lor \bot\right)$
- $p_i \lor p_j$
- $p_i \lor p_j \lor p_k$
- $p_i \lor (p_j \land p_k)$

Counter Examples

- $\cancel{p_i \land p_j}$
- $\cancel{p_i \land p_j \land p_k}$
- $\cancel{p_i \land (p_j \lor p_k)}$

Outline

Horn Clauses

Definition: Horn Clause

An implication whose premise (body) is a conjunction ($\wedge$) of positive literals and whose conclusion (head) is a single positive literals:

$$\overbrace{(b_0 \wedge b_1 \wedge \ldots \wedge b_n)}^{\text{body}} \implies \overbrace{h}^{\text{head}}$$

Equivalently, a disjunction $\vee$ with at most one positive literal:

$$\overbrace{\neg b_0 \vee \neg b_1 \vee \ldots \vee \neg b_n}^{\text{body}} \vee \overbrace{h}^{\text{head}}$$

Horn clause reasoning is more efficient than general Boolean formulae.

Many real-world domains can be expressed with only Horn clauses.

Examples: Horn Clauses

Examples

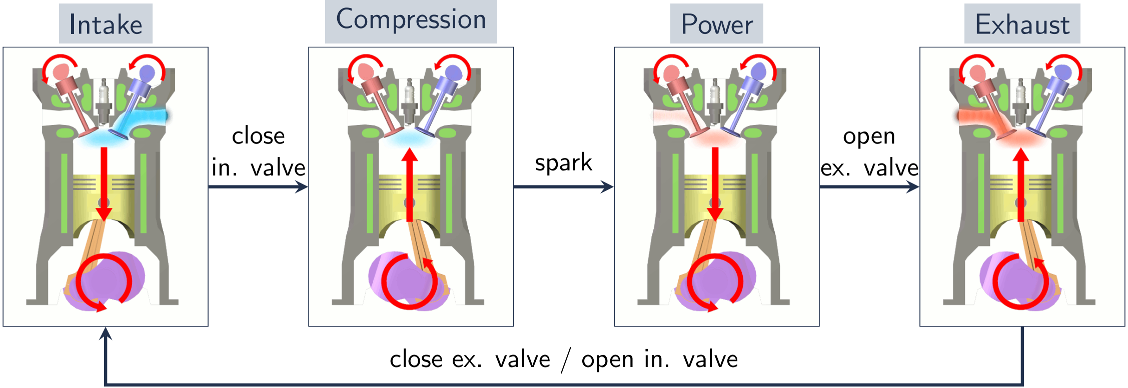
- $\left(a\right) = \left(\top \implies a\right)$
- $\left(\neg a \vee b\right) = \left(a \implies b\right)$
- $\left(\neg a \implies \neg b\right) = \left(a \vee \neg b\right)$
  $= \left(b \implies a\right)$
- $\left(\neg a \vee \neg b\right) = \left((a \wedge b) \implies \bot\right)$

Counterexamples

- $\left(a \vee b\right) = \left(\neg a \implies b\right)$
- $\left(\neg a \vee b \vee c\right) = \left(a \implies (b \vee c)\right)$

Example: Engine Troubleshooting

Four-cycle Engine Operation



https://commons.wikimedia.org/w/index.php?curid=180927

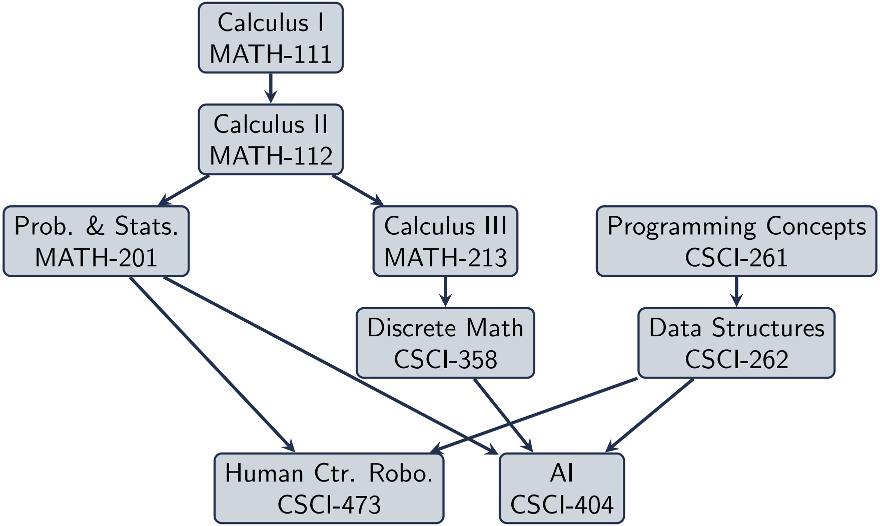
$$\texttt{ignition} \iff (\texttt{fuel} \wedge \texttt{compression} \wedge \texttt{spark})$$

Example: Engine Troubleshooting

Troubleshooting Knowledge Base

- $\texttt{ignition} \iff (\texttt{fuel} \wedge \texttt{compression} \wedge \texttt{spark})$
- $\texttt{fuel} \iff (\texttt{full-tank} \wedge \texttt{clean-carbs})$
- $\texttt{compression} \iff (\texttt{clean-air-filter} \wedge \texttt{good-piston-rings})$
- $\texttt{spark} \iff (\texttt{battery-charged} \wedge \texttt{good-connection} \wedge \texttt{good-plugs})$
- $\texttt{turns-over} \implies \texttt{battery-charged}$

Exercise: Course Prerequisites



Exercise: Course Prerequisites

continued

Overview: Forward Chaining

1. Start with known propositions, $\top \implies p$
2. Derive new propositions and add to knowledgebase
   2.1 For $(p_0 \wedge \ldots \wedge p_n) \implies p_h$
   2.2 When all $p_0 \ldots p_n$ are known true
   2.3 p_h must be true
3. Terminate when either we prove the query proposition true or we can derive no more

Example: Forward Chaining

The engine turns over but won't start. I just cleaned the carbs and filled the gas tank. Should I check the spark plugs?

$\mathtt{ignition} \iff (\mathtt{fuel} \wedge \mathtt{compression} \wedge \mathtt{spark})$	turns-over
$\mathtt{fuel} \iff (\mathtt{full\text{-}tank} \wedge \mathtt{clean\text{-}carbs})$	full-tank
$\mathtt{compression} \iff (\mathtt{clean\text{-}air\text{-}filter} \wedge \mathtt{good\text{-}piston\text{-}rings})$	clean-carbs
$\mathtt{spark} \iff (\mathtt{battery\text{-}charged} \wedge \mathtt{good\text{-}connection} \wedge \mathtt{good\text{-}plugs})$	
$\mathtt{turns\text{-}over} \implies \mathtt{battery\text{-}charged}$	

$\mathtt{ignition} \iff (\mathtt{fuel} \wedge \mathtt{compression} \wedge \mathtt{spark})$	turns-over
$\mathtt{fuel} \iff \left(\cancelto{\top}{\mathtt{full\text{-}tank}} \wedge \cancelto{\top}{\mathtt{clean\text{-}carbs}}\right)$	full-tank
$\mathtt{compression} \iff (\mathtt{clean\text{-}air\text{-}filter} \wedge \mathtt{good\text{-}piston\text{-}rings})$	clean-carbs
$\mathtt{spark} \iff (\mathtt{battery\text{-}charged} \wedge \mathtt{good\text{-}connection} \wedge \mathtt{good\text{-}plugs})$	
$\cancelto{\top}{\mathtt{turns\text{-}over}} \implies \mathtt{battery\text{-}charged}$	

Example: Forward Chaining

continued

Rules	Facts
$\mathtt{ignition} \iff (\mathtt{fuel} \land \mathtt{compression} \land \mathtt{spark})$ $\mathtt{fuel} \iff (\cancel{\mathtt{full\text{-}tank}} \land \cancel{\mathtt{clean\text{-}carbs}})$ $\mathtt{compression} \iff (\mathtt{clean\text{-}air\text{-}filter} \land \mathtt{good\text{-}piston\text{-}rings})$ $\mathtt{spark} \iff (\mathtt{battery\text{-}charged} \land \mathtt{good\text{-}connection} \land \mathtt{good\text{-}plugs})$ $\cancel{\mathtt{turns\text{-}over}} \implies \mathtt{battery\text{-}charged}$	turns-over full-tank clean-carbs fuel battery-charged
$\mathtt{ignition} \iff \left(\cancel{\mathtt{fuel}}^{\top} \land \mathtt{compression} \land \mathtt{spark}\right)$ $\mathtt{fuel} \iff (\cancel{\mathtt{full\text{-}tank}} \land \cancel{\mathtt{clean\text{-}carbs}})$ $\mathtt{compression} \iff (\mathtt{clean\text{-}air\text{-}filter} \land \mathtt{good\text{-}piston\text{-}rings})$ $\mathtt{spark} \iff \left(\cancel{\mathtt{battery\text{-}charged}}^{\top} \land \mathtt{good\text{-}connection} \land \mathtt{good\text{-}plugs}\right)$ $\cancel{\mathtt{turns\text{-}over}} \implies \mathtt{battery\text{-}charged}$	turns-over full-tank clean-carbs fuel battery-charged

Algorithm: Forward Chaining

Procedure forward-chain(K,q)

```
W ← {c ∈ K | c is a literal};
V ← ∅;
while ¬empty(W) do
    let x = pop(W) in
        if head(c) = q then return ⊤;
        else if x ∉ V then
            V ← V ∪ {x};
            foreach c ∈ K where x ∈ body(c) do
                Remove x from body(c);
                if empty(body(c)) then
                    W ← push(head(c), W)
return ⊥;
```

Efficiency Notes

- Index K by variables in its body
- Track remaining (unproven) terms in body with count
  - Initialize count to length of body
  - Decrement count each time we remove a term
  - When 0, head is true

Exercise: Forward Chaining

I want to take Human-Centered Robotics (CSCI-473). Do I need Calculus III (MATH-213)?

Exercise: Forward Chaining

Exercise: Forward Chaining

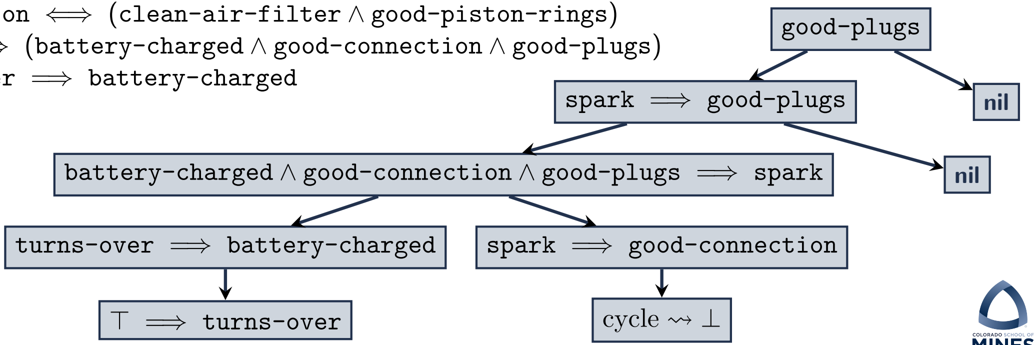
Overview: Backward Chaining

1. Start with query proposition q
2. Recursively follow clauses that imply q:
   2.1 $p_0 \wedge \ldots \wedge p_n \implies q$
   2.2 $\ell_0 \wedge \ldots \wedge \ell_n \implies p_0$
   2.3 etc.
3. Terminate recursion when:
   3.1 we arrive at a know proposition ($\top$)
   3.2 or there are no more clauses that imply the current proposition ($\bot$)

Example: Backward Chaining

The engine turns over but won't start. I just cleaned the carbs and filled the gas tank. Should I check the spark plugs?

$$\texttt{ignition} \iff (\texttt{fuel} \land \texttt{compression} \land \texttt{spark})$$
$$\texttt{fuel} \iff (\texttt{full-tank} \land \texttt{clean-carbs})$$
$$\texttt{compression} \iff (\texttt{clean-air-filter} \land \texttt{good-piston-rings})$$
$$\texttt{spark} \iff (\texttt{battery-charged} \land \texttt{good-connection} \land \texttt{good-plugs})$$
$$\texttt{turns-over} \implies \texttt{battery-charged}$$



Algorithm: Backward Chaining

Simple

Procedure backward-chain(K,q)

function $visit(V, x)$ **is**
    if $x \in V$ **then** **return** $\bot$; // circular definition
    else // $\exists c \in K,\ ((\mathtt{head}(c) = x) \land (\mathtt{body}(c)))$
        foreach $c \in K$ **where** $x = \mathtt{head}(c)$ **do**
            foreach $b \in \mathtt{body}(c)$ **do** // $\forall b \in \mathtt{body}(c),\ b$
                if $\neg\mathtt{visit}(V \cup \{x\},\ b)$ **then** **return** $\bot$;
            return $\top$;
        return $\bot$;

return $\mathtt{visit}(\emptyset, q)$;

Algorithm: Backward Chaining

Memoizing

Procedure backward-chain(K,q)

```
M ← ∅; // Cached set of true propositions
function visit(V, x) is
    if x ∈ M then return ⊤; // cached result
    else if x ∈ V then return ⊥; // circular definition
    else // ∃c ∈ K, ((head(c) = x) ∧ (body(c)))
        foreach c ∈ K where x = head(c) do
            foreach b ∈ body(c) do // ∀b ∈ body(c), b
                if ¬visit(V ∪ {x}, b) then return ⊥;
            M ← M ∪ {x};
            return ⊤;
        return ⊥;
return visit(∅, q);
```

Exercise: Backward Chaining

I want to take Human-Centered Robotics (CSCI-473). Do I need Calculus III (MATH-213)?

Knowledge Base Indexing

Procedure forward-chain(K,q)

```
W ← {c ∈ K | c is a literal};
V ← ∅;
while ¬empty(W) do
    let x = pop(W) in
        if head(c) = q then return ⊤;
        else if x ∉ V then
            V ← V ∪ {x};
            /* Naive: O(n) to select matching clauses */
            foreach c ∈ K where x ∈ body(c) do
                Remove x from body(c);
                if empty(body(c)) then
                    W ← push(head(c), W)
return ⊥;
```

Procedure backward-chain(K,q)

```
M ← ∅;
function visit(V, x) is
    if x ∈ M then return ⊤;
    else if x ∈ V then return ⊥;
    else
        /* Naive: O(n) to select matching clauses */
        foreach c ∈ K where x = head(c) do
            foreach b ∈ body(c) do
                if ¬visit(V ∪ {x}, b) then
                    return ⊥;
            M ← M ∪ {x};
            return ⊤;
        return ⊥;
return visit(∅, q);
```

Exercise: Knowledge base Indexing

Forward Chaining

foreach $c \in K$ **where** $x \in \texttt{body}(c)$

Procedure forward-chain-index(K)

Backward Chaining

foreach $c \in K$ **where** $x = \texttt{head}(c)$

Procedure backward-chain-index(K)

Outline

Boolean Logic

Forward and Backward Chaining
- Horn Clauses
- Forward Chaining
- Backward Chaining

Satisfiability
- Conjunctive Normal Form
- Davis-Putnam-Logemann-Loveland

Tools
- Logic Programming
- Constraint Solvers

SAT Problem

Given: A Boolean formula:

- Variables $P = p_1 \ldots p_n$
- Formula $\phi : \mathbb{B}^n \mapsto \mathbb{B}$

Find: Is $\phi(P)$ satisfiable?

- $\exists P,\ (\phi(P) = 1)$
- What is P?

Solution: Davis-Putnam-Logeman-Loveland (DPLL)
Backtracking Search

General and efficient reasoning over propositional logic.

So what?

- Software verification
- AI / Robot Planning
- Combinatorial Design

Conjunctive Normal Form

S-Expression

Definition (Conjunctive Normal Form)

A conjunction of disjunction of literals:

$$
\begin{array}{ll}
(\texttt{AND} & (\texttt{OR}\ \ell_{0,0}\ \ell_{0,1}\ \ \ldots\ \ \ell_{0,n}) \\
 & (\texttt{OR}\ \ell_{1,0}\ \ell_{1,1}\ \ \ldots\ \ \ell_{1,n}) \\
 & \ldots \\
 & (\texttt{OR}\ \ell_{n,0}\ \ell_{n,1}\ \ \ldots\ \ \ell_{n,n}))
\end{array}
$$

where each $\ell_{i,j}$ is a literal, that is one of p, $(\texttt{NOT}\ p)$.

Conjunctive Normal Form

infix

Definition (Conjunctive Normal Form)

A conjunction of disjunction of literals

Examples

- $p_i \rightsquigarrow$ (AND (OR p_i))
- $\neg p_i \lor p_j \rightsquigarrow$ (AND (OR (NOT p_i) p_j))
- $p_i \land (p_j \lor p_k)$
- $(p_i \lor p_j) \land (\neg p_i \lor p_k)$

Counter Examples

- $\cancel{p_i \lor (p_j \land p_k)}$
- $\cancel{\neg(p_i \lor p_j)}$
- $\cancel{p_i \implies p_j}$

Conversion to CNF

1. **Eliminate** $\iff$**:** $\left(\alpha \iff \beta\right) \rightsquigarrow \left((\alpha \implies \beta) \wedge (\beta \implies \alpha)\right)$
2. **Eliminate** $\implies$**:** $\left(\alpha \implies \beta\right) \rightsquigarrow \left(\neg\alpha \vee \beta\right)$
3. **Eliminate** $\oplus$**:** $\left(a \oplus b\right) \rightsquigarrow \left((a \vee b) \wedge \neg(a \wedge b)\right)$
4. **Move in** $\neg$**:**
   - $\left(\neg(\neg\alpha)\right) \rightsquigarrow \left(\alpha\right)$
   - $\left(\neg(\alpha \wedge \beta)\right) \rightsquigarrow \left((\neg\alpha \vee \neg\beta)\right)$
   - $\left(\neg(\alpha \vee \beta)\right) \rightsquigarrow \left((\neg\alpha \wedge \neg\beta)\right)$
5. **Distribute** $\vee$ **over** $\wedge$: $\left(\alpha \vee (\beta \wedge \gamma)\right) \rightsquigarrow \left((\alpha \vee \beta) \wedge (\alpha \vee \gamma)\right)$

Example: CNF Conversions

- $\left(a \iff b\right) \overset{\text{Elim.} \iff}{\rightsquigarrow} \left((a \implies b) \land (b \implies a)\right) \overset{\text{Elim.} \implies}{\rightsquigarrow} \left((\neg a \lor b) \land (\neg b \lor a)\right)$
- $\left(a \implies \neg(b \lor c)\right) \overset{\text{Elim.} \implies}{\rightsquigarrow} \left(\neg a \lor \neg(b \lor c)\right) \overset{\text{move} \neg}{\rightsquigarrow} \left(\neg a \lor (\neg b \land \neg c)\right)$
  $\overset{\text{dist.} \lor}{\rightsquigarrow} \left((\neg a \lor \neg b) \land (\neg a \lor \neg c)\right)$

Exercise: CNF Conversions 0

- $\left(\neg(a \lor b) \implies c\right)$
- $\left(\neg(a \land b) \lor (a \land c)\right)$

Exercise: CNF Conversions 1

- $\left(\neg(a \lor b) \oplus (a \land c)\right)$

DPLL Outline

For ϕ in conjunctive normal form:

Base Case:
1. If ϕ has all true clauses ($\forall$), return true.
2. If ϕ has any false clauses ($\exists$), return false.

Recursive Case:
1. Propagate values from unit (single-variable) clauses.
2. Choose a branching variable v.
3. Branch (recurse) for $v = 1$ or $v = 0$.

Unit Propagation

Procedure unit-propagate(ϕ)

```
if φ has some unit clause with variable v then
    φ′ ← replace v in φ with value to make unit clause true;
    return unit-propagate(φ);
else
    return φ;
```

Example: Unit Propagation

- $\left(\overbrace{a}^{\textit{unit}} \land (a \lor b) \land (\neg b \lor c)\right) \overset{a=1}{\rightsquigarrow} \left(1 \land \cancelto{1}{(1 \lor b)} \land (\neg b \lor c)\right) \overset{\textit{simplify}}{\rightsquigarrow} \left((\neg b \lor c)\right)$

Exercise: Unit Propagation

- $\left(\overbrace{\neg a}^{unit} \land (a \lor b \lor c)\right)$
- $\Big((a \lor \neg b) \land b \land (b \lor c)\Big)$
- $\Big((a \lor \neg b \lor \neg c) \land (b \lor \neg c) \land c\Big)$
- $\Big(a \land (\neg a \lor b) \land (\neg a \lor \neg b)\Big)$

DPLL Algorithm

Procedure DPLL(ϕ)

```
φ' ← unit-propagate(φ);
if φ' = true then
    return true;
else if φ' = false then
    return false;
else // Recursive case
    v ← choose-variable(φ');
    if DPLL(φ' ∧ v) then
        return true;
    else
        return DPLL(φ' ∧ ¬v);
```

Example 0: DPLL

$$\boxed{(a \lor \neg b) \land (\neg a \lor b)}$$

$$\Big\downarrow \text{ choose } a = 1$$

$$\boxed{\cancelto{1}{(1 \lor \neg b)} \land (\cancelto{0}{\neg 1} \lor b)}$$

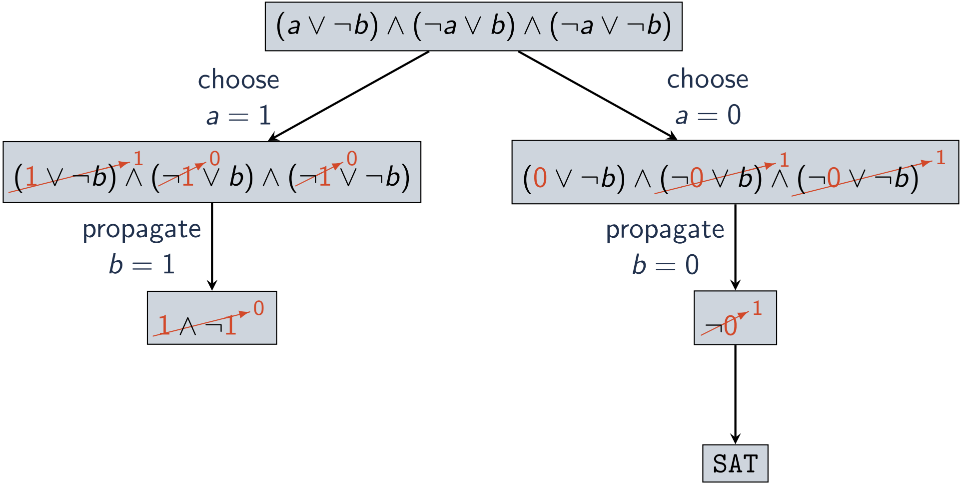
$$\Big\downarrow \text{ propagate } b = 1$$

$$\boxed{1}$$

$$\Big\downarrow$$

$$\boxed{\texttt{SAT}}$$

Example 1: DPLL



Exercise 0: DPLL

Exercise 1: DPLL

Exercise 2: DPLL

Exercise 3: DPLL

Exercise 4: DPLL

Outline

Logic Programming and Constraint Solvers

Logic Programming

Statements:
- Logical Expressions
- Query

Execution: Logical Inference

Output: Query true/false

Constraint Solvers

Algorithm: DPLL + heuristics

Input:
- Set of variables
- Constraint equations (assertions)

Output: Satisfying variable assignment

Different view, similar operation and capabilities

Prolog

- Statements are Horn clauses

  Horn: $\text{body} \implies \text{head}$

  Prolog: `head :- body`

- Example:

  Horn: $(a \wedge b) \implies c$

  Prolog: `c :- a, b.`

- Evaluation:
  - Query: `?x`
  - Backward Chaining

Answer-Set Programming

- Horn Clauses:

  Horn: $(a \wedge b) \implies c$

  ASP: `c :- a, b.`

- Choice Rule:

  Math: $p \implies (s \vee t)$

  ASP: `{s,t} :- p.`

- Constraint:

  Math: $\Big((s \wedge \neg t) \implies \bot\Big) = \Big(\neg(s \wedge \neg t)\Big) = \Big(\neg s \vee t\Big)$

  ASP: `:- s, not t.`

- Evaluation: DPLL

Satisfiability Solvers

Math

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

DIMACS

```
c Problem Definition (p):
c    * problem type (cnf)
c    * variable count (3)
c    * clause count (2)

p cnf 3 2
1 2 0
1 -3 0
```

Output

```
s SATISFIABLE
v 1 2 -3 0
```

http://www.satcompetition.org/

Satisfiability Modulo Theories

Math

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

(SMT also handles non-Boolean types.)

SMTLib

```
(declare-fun a () Bool)
(declare-fun b () Bool)
(declare-fun c () Bool)

(assert (or a b))
(assert (or a (not c)))

(check-sat)

(get-value (a b c))
```

Output

```
sat
((a false)
 (b true)
 (c false))
```

http://smtlib.cs.uiowa.edu/ http://www.smtcomp.org/