Propositional Calculus

Dr. Neil T. Dantam

CSCI-534, Colorado School of Mines

Spring 2020

MINES

Calculus?

Definition (Calculus)

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

Example

- 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.

MINES

Introduction

Definition (Propositional Calculus)

A calculus over propositions—variables that may be true or false.
Also called: propositional logic, zeroth-order logic, or Boolean Algebra

Why?

- Propositional logic is foundational
- Many hard problems can be **reduced** to propositional logic

Outcomes

- Understand Boolean operators
- Create logical models of real-world problems
- Implement logical inference via:
  - Forward Chaining
  - Backward Chaining
  - DPLL (backtracking search)

MINES

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

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

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

Implies

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

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

Biconditional (iff)

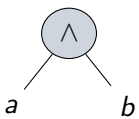
$$(a \iff b) \triangleq (a \implies b) \wedge (b \implies a)$$
$$\triangleq \neg(a \oplus b)$$
$$\triangleq (a \wedge b) \vee (\neg a \wedge \neg b)$$

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

Example: Boolean Formulae as S-expressions

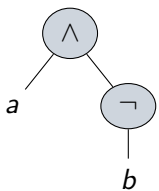
$a \wedge b$

```
(and a b)
```



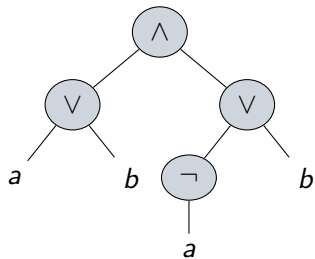
$a \wedge \neg b$

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



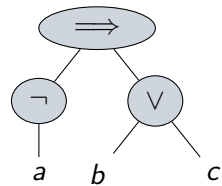
$(a \vee b) \wedge (\neg a \vee b)$

```
(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

$(a \wedge b) \implies c$

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

$\neg a \vee \neg b \vee c$

All these formulae are equivalent

N-ary Boolean Operators

AND

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

OR

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

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

$\chi = \top$

$$
\begin{aligned}
&(\texttt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \alpha_n)\\
\rightsquigarrow &\left(\texttt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \cancel{\alpha_n}\right)^{\top}\\
\rightsquigarrow &(\texttt{AND}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1})\\
\rightsquigarrow &\left(\texttt{AND}\ \cancel{\alpha_0}^{\top}\ \cancel{\alpha_1}^{\top}\ \ldots\ \cancel{\alpha_{n01}}\right)^{\top}\\
\rightsquigarrow &(\texttt{AND})\\
\rightsquigarrow &\top
\end{aligned}
$$

OR

$\chi = \bot$

$$
\begin{aligned}
&(\texttt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \alpha_n)\\
\rightsquigarrow &\left(\texttt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1}\ \cancel{\alpha_n}\right)^{\bot}\\
\rightsquigarrow &(\texttt{OR}\ \alpha_0\ \alpha_1\ \ldots\ \alpha_{n-1})\\
\rightsquigarrow &\left(\texttt{OR}\ \cancel{\alpha_0}^{\bot}\ \cancel{\alpha_1}^{\bot}\ \ldots\ \cancel{\alpha_{n01}}\right)^{\bot}\\
\rightsquigarrow &(\texttt{OR})\\
\rightsquigarrow &\bot
\end{aligned}
$$

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: $\neg p$

Example (Literals)

- p_i
- $\neg p_i$

Counterexample (Not Literals)

- ~~$p_i \wedge p_j$~~
- ~~$p_i \vee p_j$~~
- ~~$\neg(p_i \wedge p_j)$~~

Definitions

Conjunction

Definition (Conjunction)

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

Example (Conjunctions)

- $p_i \rightsquigarrow (\texttt{AND}\ p_i)$
- $p_i \wedge p_j \rightsquigarrow (\texttt{AND}\ p_i\ p_j)$
- $p_i \wedge p_j \wedge p_k \rightsquigarrow (\texttt{AND}\ p_i\ p_j\ p_k)$
- $p_i \wedge (p_j \vee p_k) \rightsquigarrow (\texttt{AND}\ p_i\ (\texttt{OR}\ p_j\ p_k))$

Example (Not Conjunctions)

- ~~$p_i \vee p_j$~~
- ~~$p_i \vee (p_j \wedge p_k)$~~
- ~~$(p_i \wedge p_j) \vee p_k$~~

Definitions

Disjunction

Definition (Disjunction)

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

Example (Disjunctions)

- $p_i \rightsquigarrow (\texttt{OR}\ p_i)$
- $p_i \vee p_j \rightsquigarrow (\texttt{OR}\ p_i\ p_j)$
- $p_i \vee p_j \vee p_k \rightsquigarrow (\texttt{OR}\ p_i\ p_j\ p_k)$
- $p_i \vee (p_j \wedge p_k) \rightsquigarrow (\texttt{OR}\ p_i\ (\texttt{AND}\ p_j\ p_k))$

Example (Not Disjunctions)

- ~~$p_i \wedge p_j$~~
- ~~$p_i \wedge p_j \wedge p_k$~~
- ~~$p_i \wedge (p_j \vee p_k)$~~

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

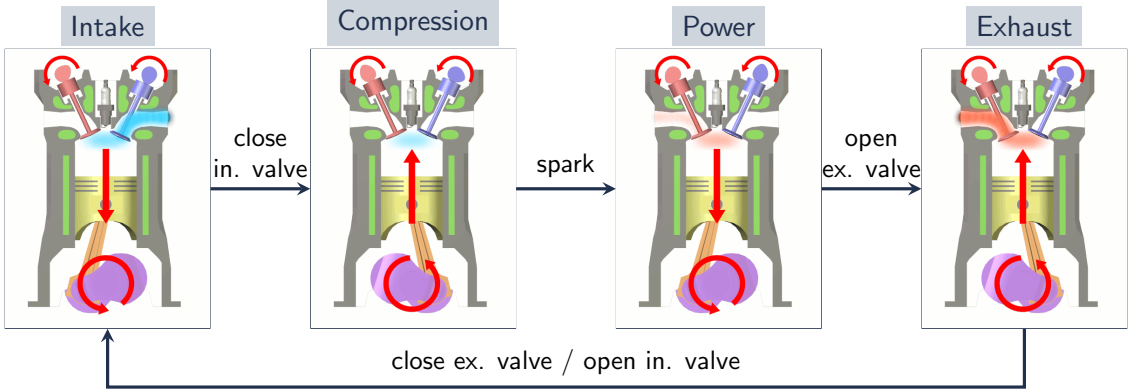
- $\Big(a\Big) = \Big(\top \implies a\Big)$
- $\Big(\neg a \vee b\Big) = \Big(a \implies b\Big)$
- $\Big(\neg a \implies \neg b\Big) = \Big(a \vee \neg b\Big)$
  $= \Big(b \implies a\Big)$
- $\Big(\neg a \vee \neg b\Big) = \Big((a \wedge b) \implies \bot\Big)$

Counterexamples

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

Example: Engine Troubleshooting

Four-cycle Engine Operation



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

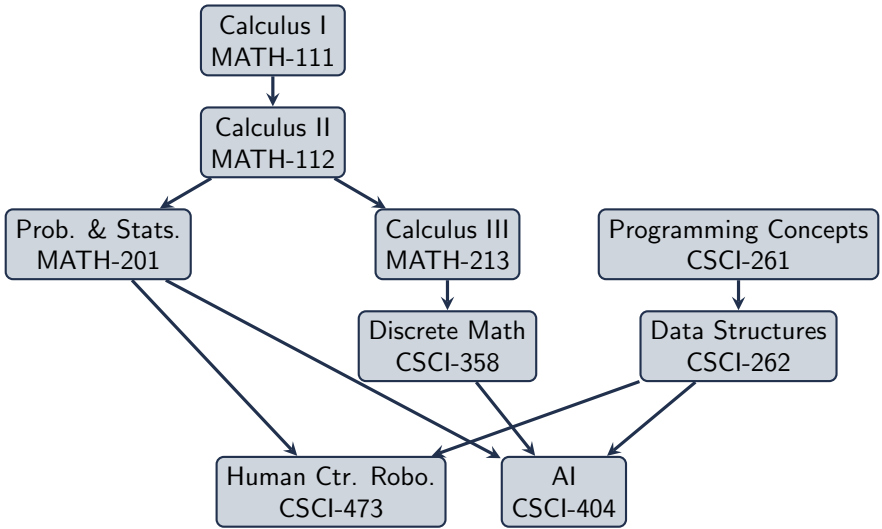
$$\texttt{ignition} \iff (\texttt{fuel} \wedge \texttt{compression} \wedge \texttt{spark})$$

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

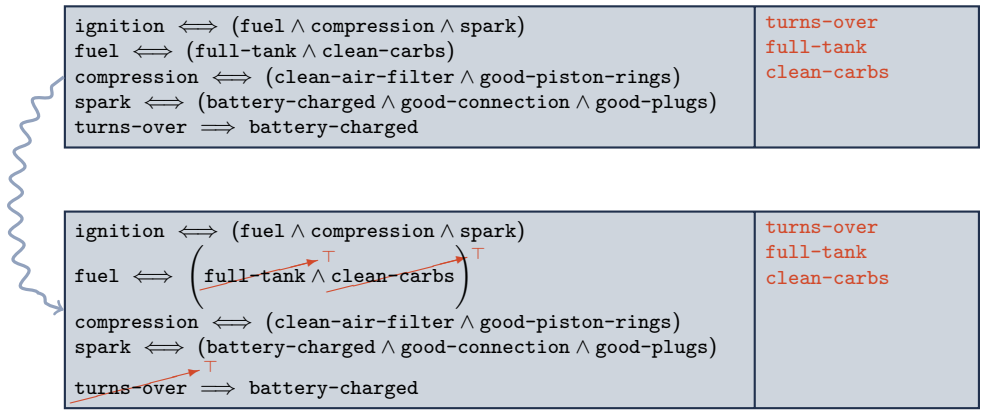
continued

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

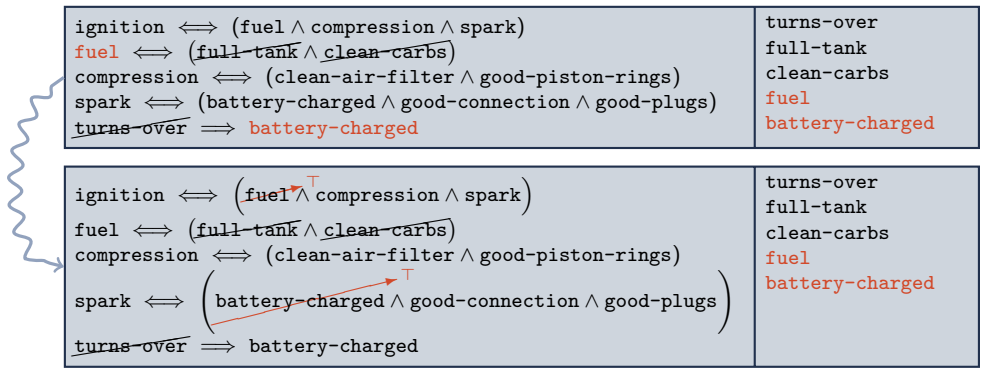
Example: Forward Chaining

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



Example: Forward Chaining

continued



Algorithm: Forward Chaining

Procedure forward-chain(K,q)

```
1  W ← {c ∈ K | c is a literal};
2  V ← ∅;
3  while ¬empty(W) do
4      let x ← pop(W) in
5          if x ∉ V then
6              if x = q then return ⊤;
7              V ← V ∪ {x};
8              foreach c ∈ K where x ∈ body(c) do
9                  Remove x from body(c);
10                 if empty(body(c)) then
11                     W ← push(head(c), W)
12 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

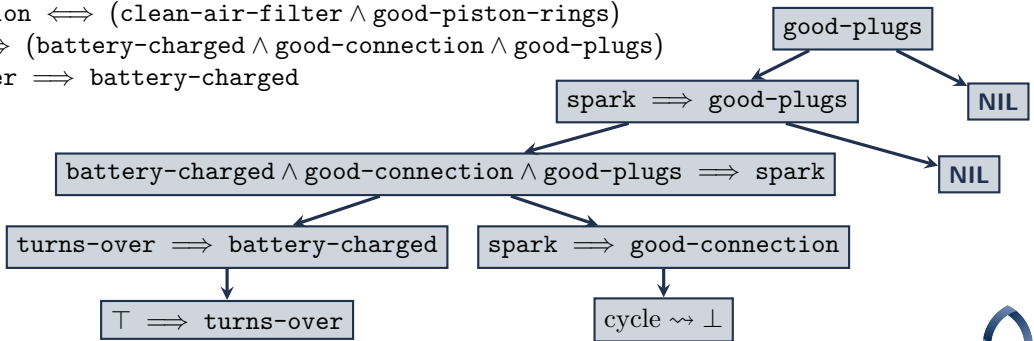
1. Start with query proposition q
2. Recursively follow clauses that imply q:
   - 2.1 $p_0 \wedge \ldots \wedge p_n \implies q$
   - 2.2 $\ell_0 \wedge \ldots \wedge \ell_n \implies p_0$
   - 2.3 etc.
3. Terminate recursion when:
   - 3.1 we arrive at a known proposition ($\top$)
   - 3.2 or there are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

MINES

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?

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



Algorithm: Backward Chaining

Simple

Procedure backward-chain(K,q)

```
1 function visit(V, x) is
2     if x ∈ V then return ⊥; // circular definition
3     else
          /* Some clause c in K implies x                    */
          /* ∃c ∈ K, ((head(c) = x) ∧ (body(c)))             */
4         foreach c ∈ K where x = head(c) do
5             if ∀p ∈ body(c), visit(V ∪ {x}, p) then
                  /* Every body proposition p in c is true  */
6                 return ⊤;
7         return ⊥;
8 return visit(∅, q);
```

Algorithm: Backward Chaining

Memoizing

Procedure backward-chain(K,q)

```
1  M ← ∅; // Cached set of true propositions
2  function visit(V, x) is
3      if x ∈ M then return ⊤; // cached result
4      else if x ∈ V then return ⊥; // circular definition
5      else // ∃c ∈ K, ((head(c) = x) ∧ (body(c)))
6          foreach c ∈ K where x = head(c) do
7              if ∀p ∈ body(c), visit(V ∪ {x}, p) then
8                  M ← M ∪ {x};
9                  return ⊤;
10             return ⊤;
11         return ⊥;
12 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)

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

Procedure backward-chain(K,q)

```
1  M ← ∅; // Cached set of true propositions
2  function visit(V, x) is
3      if x ∈ M then return ⊤; // cached result
4      else if x ∈ V then return ⊥; // circular definition
5      else // ∃c ∈ K, ((head(c) = x) ∧ (body(c)))
           // Naive: O(n) to select matching clauses
6          foreach c ∈ K where x = head(c) do
7              if ∀p ∈ body(c), visit(V ∪ {x}, p) then
8                  M ← M ∪ {x};
9                  return ⊤;
10             return ⊤;
11         return ⊥;
12 return visit(∅, q);
```

Exercise: Knowledge base Indexing

Forward Chaining

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

Procedure forward-chain-index(K)

1
2
3
6

Backward Chaining

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

Procedure backward-chain-index(K)

1

Outline

MINES

SAT Problem

Given: A Boolean formula:

- Variables $P = p_1 \dots p_n$
- 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.

MINES

So what?

- Software verification
- AI / Robot Planning
- Combinatorial Design

Conjunctive Normal Form

S-Expression

Definition (Conjunctive Normal Form)

A conjunction of disjunction of literals:

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

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

- $p_i \rightsquigarrow (\texttt{AND}\ (\texttt{OR}\ p_i))$
- $\neg p_i \vee p_j \rightsquigarrow$ $(\texttt{AND}\ (\texttt{OR}\ (\texttt{NOT}\ p_i)\ p_j))$
- $p_i \wedge (p_j \vee p_k)$
- $(p_i \vee p_j) \wedge (\neg p_i \vee p_k)$

Counter Examples

- ~~$p_i \vee (p_j \wedge p_k)$~~
- ~~$\neg(p_i \vee p_j)$~~
- ~~$p_i \implies p_j$~~

Conversion to CNF

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

Example: CNF Conversions

- $\Big(a \iff b\Big) \overset{\text{Elim.} \iff}{\rightsquigarrow} \Big((a \implies b) \wedge (b \implies a)\Big) \overset{\text{Elim.} \implies}{\rightsquigarrow} \Big((\neg a \vee b) \wedge (\neg b \vee a)\Big)$
- $\Big(a \implies \neg(b \vee c)\Big) \overset{\text{Elim.} \implies}{\rightsquigarrow} \Big(\neg a \vee \neg(b \vee c)\Big) \overset{\text{move} \neg}{\rightsquigarrow} \Big(\neg a \vee (\neg b \wedge \neg c)\Big)$
  $\overset{\text{dist.} \vee}{\rightsquigarrow} \Big((\neg a \vee \neg b) \wedge (\neg a \vee \neg c)\Big)$

Exercise: CNF Conversions 0

- $\Big(\neg(a \vee b) \implies c\Big)$
- $\Big(\neg(a \wedge b) \vee (a \wedge c)\Big)$

Exercise: CNF Conversions 1

- $\Big(\neg(a \vee b) \oplus (a \wedge c)\Big)$

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)

```
1 if φ has some unit clause with variable v then
2     let φ' ← replace v in φ with value to make unit clause true in
3         return unit-propagate(φ');
4 else
5     return φ;
```

Example: Unit Propagation

- $\Big(\overbrace{a}^{unit} \wedge (a \vee b) \wedge (\neg b \vee c)\Big) \overset{a=1}{\rightsquigarrow} \Big(1 \wedge (\cancel{1 \vee b}) \wedge (\neg b \vee c)\Big) \overset{simplify}{\rightsquigarrow} \Big((\neg b \vee c)\Big)$

Exercise: Unit Propagation

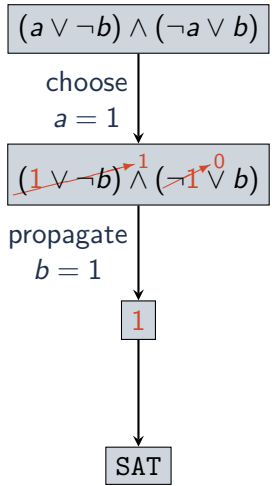
- $\Big(\overbrace{\neg a}^{unit} \wedge (a \vee b \vee c) \Big)$
- $\Big((a \vee \neg b) \wedge b \wedge (b \vee c) \Big)$
- $\Big((a \vee \neg b \vee \neg c) \wedge (b \vee \neg c) \wedge c \Big)$
- $\Big(a \wedge (\neg a \vee b) \wedge (\neg a \vee \neg b) \Big)$

DPLL Algorithm

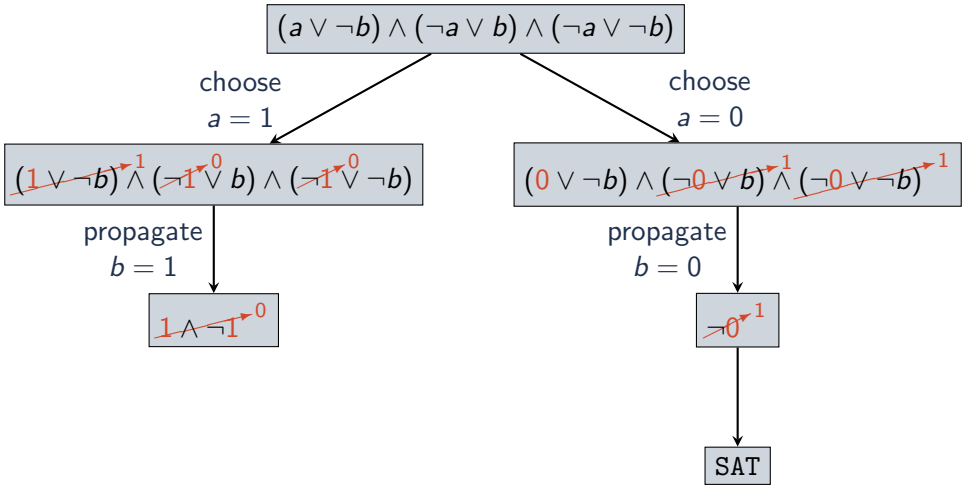
Procedure DPLL(ϕ)

```
1  let φ' ← unit-propagate (φ) in
2      if φ' = ⊤ then
3          return ⊤;
4      else if φ' = ⊥ then
5          return ⊥;
6      else // Recursive case
7          let v ← choose-variable (φ') in
8              if DPLL(φ' ∧ v) then
9                  return true;
10             else
11                 return DPLL(φ' ∧ ¬v);
```

Example 0: DPLL



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

  ASP: `:- s, not t.`

- Evaluation: DPLL

Satisfiability Solvers

Math

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

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

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

(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/

References

Textbook: Russell & Norvig.

- Ch 7.4 Propositional Logic
- Ch 7.5 Propositional Theorem Proving
- Ch 7.6 Effective Propositional Model Checking

Further Reading:

- **Conflict-Driven Clause Learning:** identifies conflicts before backtracking and avoids in future branches
- De Moura, L. and Bjørner, N., 2011. **Satisfiability modulo theories**: introduction and applications. Communications of the ACM.

MINES