IA008: Computational Logic

1. Propositional Logic

Achim Blumensath
blumens@fi.muni.cz

Faculty of Informatics, Masaryk University, Brno

Basic Concepts

Propositional Logic

Syntax

- Variables $A, B, C, \ldots, X, Y, Z, \ldots$
- Operators $\wedge, \vee, \neg, \rightarrow, \leftrightarrow$

Semantics

$$\mathfrak{I} \models \varphi \qquad\qquad \mathfrak{I} : \text{Variables} \rightarrow \{\text{true}, \text{false}\}$$

Examples

$$\varphi := A \wedge (A \rightarrow B) \rightarrow B\,,$$
$$\psi := \neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)\,.$$

Terminology

- **entailment** $\varphi \vDash \psi$ (do not confuse with $\mathfrak{I} \vDash \varphi$!)
- **equivalence** $\varphi \equiv \psi$ (do not confuse with $\varphi = \psi$!)
- $\varphi \equiv \psi$ iff $\varphi \vDash \psi$ and $\psi \vDash \varphi$

Terminology

- **entailment** $\varphi \vDash \psi$ (do not confuse with $\mathfrak{I} \vDash \varphi$!)
- **equivalence** $\varphi \equiv \psi$ (do not confuse with $\varphi = \psi$!)
- $\varphi \equiv \psi$ iff $\varphi \vDash \psi$ and $\psi \vDash \varphi$

- **satisfiability** $\varphi \not\equiv \text{false}$
- **validity** $\varphi \equiv \text{true}$
- Every valid formula is satisfiable.
- φ is valid iff $\neg\varphi$ is not satisfiable.
- $\varphi \vDash \psi$ iff $\varphi \to \psi$ is valid.

Examples

- $A \wedge (A \to B) \to B$ is

Terminology

- **entailment** $\varphi \models \psi$ (do not confuse with $\mathfrak{I} \models \varphi$!)
- **equivalence** $\varphi \equiv \psi$ (do not confuse with $\varphi = \psi$!)
- $\varphi \equiv \psi$ iff $\varphi \models \psi$ and $\psi \models \varphi$

- **satisfiability** $\varphi \not\equiv \text{false}$
- **validity** $\varphi \equiv \text{true}$
- Every valid formula is satisfiable.
- φ is valid iff $\neg\varphi$ is not satisfiable.
- $\varphi \models \psi$ iff $\varphi \rightarrow \psi$ is valid.

Examples

- $A \wedge (A \rightarrow B) \rightarrow B$ is **valid.**
- $A \vee B$ is

Terminology

- **entailment** $\varphi \vDash \psi$ (do not confuse with $\mathfrak{I} \vDash \varphi$!)
- **equivalence** $\varphi \equiv \psi$ (do not confuse with $\varphi = \psi$!)
- $\varphi \equiv \psi$ iff $\varphi \vDash \psi$ and $\psi \vDash \varphi$
- **satisfiability** $\varphi \not\equiv \text{false}$
- **validity** $\varphi \equiv \text{true}$
- Every valid formula is satisfiable.
- φ is valid iff $\neg\varphi$ is not satisfiable.
- $\varphi \vDash \psi$ iff $\varphi \to \psi$ is valid.

Examples

- $A \wedge (A \to B) \to B$ is **valid.**
- $A \vee B$ is **satisfiable** but not **valid.**
- $\neg A \wedge A$ is

Terminology

- **entailment** $\varphi \vDash \psi$ (do not confuse with $\mathfrak{I} \vDash \varphi$!)
- **equivalence** $\varphi \equiv \psi$ (do not confuse with $\varphi = \psi$!)
- $\varphi \equiv \psi$ iff $\varphi \vDash \psi$ and $\psi \vDash \varphi$
- **satisfiability** $\varphi \not\equiv \text{false}$
- **validity** $\varphi \equiv \text{true}$
- Every valid formula is satisfiable.
- φ is valid iff $\neg\varphi$ is not satisfiable.
- $\varphi \vDash \psi$ iff $\varphi \rightarrow \psi$ is valid.

Examples

- $A \wedge (A \rightarrow B) \rightarrow B$ is **valid.**
- $A \vee B$ is **satisfiable** but not **valid.**
- $\neg A \wedge A$ is **not satisfiable.**

Equivalence Transformations

De Morgan's laws

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$
$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$

Equivalence Transformations

De Morgan's laws

$$\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$$
$$\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$$

Distributive laws

$$\varphi \wedge (\psi \vee \vartheta) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \vartheta)$$
$$\varphi \vee (\psi \wedge \vartheta) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \vartheta)$$

Normal Forms

Conjunctive Normal Form (CNF)

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

Disjunctive Normal Form (DNF)

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

Clauses

Definitions

- **literal** A or $\neg A$
- **clause** set of literals $\{A, B, \neg C\}$
  short-hand for disjunction $A \vee B \vee \neg C$

Clauses

Definitions

- **literal** A or $\neg A$
- **clause** set of literals $\{A, B, \neg C\}$
  short-hand for disjunction $A \vee B \vee \neg C$

Example

CNF $\varphi := (A \vee \neg B \vee C) \wedge (\neg A \vee C) \wedge B$

clauses $\{A, \neg B, C\},\ \{\neg A, C\},\ \{B\}$

Clauses

Definitions

- **literal** A or $\neg A$
- **clause** set of literals $\{A, B, \neg C\}$
  short-hand for disjunction $A \vee B \vee \neg C$

Example

$$\text{CNF} \quad \varphi := (A \vee \neg B \vee C) \wedge (\neg A \vee C) \wedge B$$

$$\text{clauses} \quad \{A, \neg B, C\},\ \{\neg A, C\},\ \{B\}$$

Notation

$$\Phi[L := \text{true}] := \left\{\, C \setminus \{\neg L\} \;\middle|\; C \in \Phi,\ L \notin C \,\right\}.$$

The Satisfiability Problem

Davis-Putnam-Logemann-Loveland (DPLL) Algorithm

Input: a set of clauses Φ
Output: true if Φ is satisfiable, false otherwise.

```
DPLL(Φ)
  for every singleton {L} in Φ                    (* simplify Φ *)
    Φ := Φ[L := true]
  for every literal L whose negation does not occur in Φ
    Φ := Φ[L := true]
  if Φ contains the empty clause then             (* are we done? *)
    return false
  if Φ is empty then
    return true
  choose some literal L in Φ                      (* try L := true and L := false *)
  if DPLL(Φ[L := true]) then
    return true
  else
    return DPLL(Φ[L := false])
```

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: $A :=$ true

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Step 2: $B :=$ true

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: A := true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Step 2: B := true

$$\{C, D\},\ \{\neg D\},\ \{\neg C\},\ \{\neg C, \neg D\}$$

Step 3: C := false and D := false

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Step 2: $B :=$ true

$$\{C, D\},\ \{\neg D\},\ \{\neg C\},\ \{\neg C, \neg D\}$$

Step 3: $C :=$ false and $D :=$ false

$$\{D\},\ \{\neg D\}$$

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Step 2: $B :=$ true

$$\{C, D\},\ \{\neg D\},\ \{\neg C\},\ \{\neg C, \neg D\}$$

Step 3: $C :=$ false and $D :=$ false

$$\{D\},\ \{\neg D\}$$

$\varnothing$ **failure**

Example

$$\Phi := \big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Backtrack to step 2: $B :=$ false

Example

$$\Phi := \Big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\Big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Backtrack to step 2: $B :=$ false

$$\{C, D\},\ \{\neg C, \neg D\}$$

Step 3: $C :=$ true

Example

$$\Phi := \big\{\{A, B, \neg C\},\ \{\neg B, C, D\},\ \{\neg A, \neg B, \neg D\},\ \{B, C, D\},$$
$$\{\neg A, \neg B, \neg C\},\ \{\neg A, \neg C, \neg D\}\big\}$$

Step 1: $A :=$ true

$$\{\neg B, C, D\},\ \{\neg B, \neg D\},\ \{B, C, D\},\ \{\neg B, \neg C\},\ \{\neg C, \neg D\}$$

Backtrack to step 2: $B :=$ false

$$\{C, D\},\ \{\neg C, \neg D\}$$

Step 3: $C :=$ true

$\{\neg D\}$ **satisfiable**

Solution: $A =$ true, $B =$ false, $C =$ true, $D =$ false

Expressing graph problems

Vertex cover

Variables:

C_v vertex v belongs to the cover

Expressing graph problems

Vertex cover

Variables:

C_v	vertex v belongs to the cover

Formulae:

$C_u \vee C_v$	for every edge $\langle u, v \rangle \in E$
$\text{Size}_k^{\leq}$	"At most k of the C_v are true."

Expressing graph problems

Vertex cover

Variables:

C_v	vertex v belongs to the cover

Formulae:

$C_u \vee C_v$	for every edge $\langle u, v \rangle \in E$
$\text{Size}_k^{\leq}$	"At most k of the C_v are true."

Clique

Variables:

C_v	vertex v belongs to the clique

Expressing graph problems

Vertex cover

Variables:

C_v	vertex v belongs to the cover

Formulae:

$C_u \vee C_v$	for every edge $\langle u, v \rangle \in E$
$\mathrm{Size}_k^{\leq}$	"At most k of the C_v are true."

Clique

Variables:

C_v	vertex v belongs to the clique

Formulae:

$\neg C_u \vee \neg C_v$	for every non-edge $\langle u, v \rangle \notin E$
$\mathrm{Size}_k^{\geq}$	"At least k of the C_v are true."

Expressing graph problems

The $\mathrm{Size}^{\geq}_{k}$ formulae

Fix an enumeration $v_0, \ldots, v_{n-1}$ of V.

Variables:

S^k_m at least k variables C_{v_i} with $i < m$ are true

Expressing graph problems

The $\mathrm{Size}_k^{\geq}$ formulae

Fix an enumeration $v_0, \ldots, v_{n-1}$ of V.

Variables:

S_m^k at least k variables C_{v_i} with $i < m$ are true

Formulae:

$$S_m^0$$
$$\neg S_0^k \quad \text{for } k > 0$$
$$C_{v_i} \to [S_i^k \leftrightarrow S_{i+1}^{k+1}]$$
$$\neg C_{v_i} \to [S_i^k \leftrightarrow S_{i+1}^k]$$
$$S_n^k$$

Expressing graph problems

The $\text{Size}^{\geq}_{k}$ formulae

Fix an enumeration $v_0, \ldots, v_{n-1}$ of V.

Variables:

S^k_m at least k variables C_{v_i} with $i < m$ are true

Formulae:

S^0_m

$\neg S^k_0$ for $k > 0$

$C_{v_i} \to [S^k_i \leftrightarrow S^{k+1}_{i+1}]$

$\neg C_{v_i} \to [S^k_i \leftrightarrow S^k_{i+1}]$

S^k_n

		v_0		v_1		v_2	
C_{v_i}		1		0		1	
S^0_i	1		1		1		1
S^1_i	0		1		1		1
S^2_i	0		0		0		1
S^3_i	0		0		0		0

Expressing graph problems

The $\text{Size}^{\geq}_{k}$ formulae

Fix an enumeration $v_0, \ldots, v_{n-1}$ of V.

Variables:

S^k_m at least k variables C_{v_i} with $i < m$ are true

Formulae:

S^0_m

$\neg S^k_0$ for $k > 0$

$C_{v_i} \rightarrow [S^k_i \leftrightarrow S^{k+1}_{i+1}]$

$\neg C_{v_i} \rightarrow [S^k_i \leftrightarrow S^k_{i+1}]$

S^k_n

		v_0		v_1		v_2	
C_{v_i}		1		0		1	
S^0_i	1		1		1		1
S^1_i	0		1		1		1
S^2_i	0		0		0		1
S^3_i	0		0		0		0

A similar construction works for $\text{Size}^{\leq}_{k}$.

The Satisfiability Problem

Theorem

3-SAT (satisfiability for formulae in 3-CNF) is **NP-complete.**

The Satisfiability Problem

Theorem

3-SAT (satisfiability for formulae in 3-CNF) is **NP-complete.**

Proof

Given Turing machine $\mathcal{M}$ and input w, construct formula φ such that

$\mathcal{M}$ accepts w iff φ is satisfiable.

Proof

Turing machine $\mathcal{M} = \langle Q, \Sigma, \Delta, q_0, F_+, F_- \rangle$

Q	set of states
Σ	tape alphabet
Δ	set of transitions $\langle p, a, b, m, q \rangle \in Q \times \Sigma \times \Sigma \times \{-1, 0, 1\} \times Q$
q_0	initial state
F_+	accepting states
F_-	rejecting states

nondeterministic, runtime bounded by the polynomial $r(n)$

Proof

Turing machine $\mathcal{M} = \langle Q, \Sigma, \Delta, q_0, F_+, F_- \rangle$

Q	set of states
Σ	tape alphabet
Δ	set of transitions $\langle p, a, b, m, q \rangle \in Q \times \Sigma \times \Sigma \times \{-1, 0, 1\} \times Q$
q_0	initial state
F_+	accepting states
F_-	rejecting states

nondeterministic, runtime bounded by the polynomial $r(n)$

Encoding in PL

$S_{t,q}$	**state** q at time t
$H_{t,k}$	**head** in field k at time t
$W_{t,k,a}$	**letter** a in field k at time t

$$\varphi_w := \bigwedge_{t<r(n)} \left[\mathrm{ADM}_t \wedge \mathrm{INIT} \wedge \mathrm{TRANS}_t \wedge \mathrm{ACC} \right]$$

Proof

$S_{t,q}$	**state** q at time t
$H_{t,k}$	**head** in field k at time t
$W_{t,k,a}$	**letter** a in field k at time t

Admissibility formula

$$\mathrm{ADM}_t := \bigwedge_{p \neq q} [\neg S_{t,p} \vee \neg S_{t,q}] \qquad \text{unique state}$$

$$\wedge \bigwedge_{k \neq l} [\neg H_{t,k} \vee \neg H_{t,l}] \qquad \text{unique head position}$$

$$\wedge \bigwedge_{k} \bigwedge_{a \neq b} [\neg W_{t,k,a} \vee \neg W_{t,k,b}] \qquad \text{unique letter}$$

Proof

$S_{t,q}$	**state** q at time t
$H_{t,k}$	**head** in field k at time t
$W_{t,k,a}$	**letter** a in field k at time t

Initialisation formula for input: $a_0 \dots a_{n-1}$

$$\begin{aligned} \mathrm{INIT} := {} & S_{0,q_0} && \text{initial state} \\ & \wedge H_{0,0} && \text{initial head position} \\ & \wedge \bigwedge_{k<n} W_{0,k,a_k} \wedge \bigwedge_{n \leq k \leq r(n)} W_{0,k,\Box} && \text{initial tape content} \end{aligned}$$

Acceptance formula

$$\mathrm{ACC} := \bigvee_{q \in F_+} \bigvee_{t \leq r(n)} S_{t,q} \qquad \text{accepting state}$$

Proof

$S_{t,q}$	**state** q at time t
$H_{t,k}$	**head** in field k at time t
$W_{t,k,a}$	**letter** a in field k at time t

Transition formula

$$\mathrm{TRANS}_t := \bigvee_{\langle p,a,b,m,q\rangle\in\Delta} \bigvee_{k\leq r(n)} [S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \wedge H_{t+1,k+m} \wedge W_{t+1,k,b}]$$

effect of transition

$$\wedge \bigwedge_{k\leq r(n)} \bigwedge_{a\in\Sigma} [\neg H_{t,k} \wedge W_{t,k,a} \rightarrow W_{t+1,k,a}]$$

rest of tape remains unchanged

Proof

$$\mathrm{TRANS}_t := \bigvee_{\langle p,a,b,m,q\rangle\in\Delta} \bigvee_{k\le r(n)} \left[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \wedge H_{t+1,k+m} \wedge W_{t+1,k,b}\right] \wedge \ldots$$

Proof

$$\mathrm{TRANS}_t := \bigvee_{\langle p,a,b,m,q\rangle \in \Delta} \bigvee_{k \leq r(n)} \big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge$$

$$S_{t+1,q} \wedge H_{t+1,k+m} \wedge W_{t+1,k,b} \big] \wedge \ldots$$

equivalently:

$$\bigwedge_{k \leq r(n)} \bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \to \bigvee_{q \in TS(p,a)} S_{t+1,q} \Big]$$

$TS(p,a) := \{\, q \in Q \mid \langle p,a,b,m,q \rangle \in \Delta \,\}$

Proof

$$\text{TRANS}_t := \bigvee_{\langle p,a,b,m,q\rangle\in\Delta} \bigvee_{k\le r(n)} \big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \wedge H_{t+1,k+m} \wedge W_{t+1,k,b}\big] \wedge \ldots$$

equivalently:

$$\bigwedge_{k\le r(n)} \bigwedge_{p\in Q} \bigwedge_{a\in\Sigma} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \to \bigvee_{q\in TS(p,a)} S_{t+1,q}\Big]$$

$$\wedge \bigwedge_{k\le r(n)} \bigwedge_{p,q\in Q} \bigwedge_{a\in\Sigma} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \to \bigvee_{m\in TH(p,a,q)} H_{t+1,k+m}\Big]$$

$TH(p,a,q) := \{\, m \mid \langle p,a,b,m,q\rangle \in \Delta \,\}$

Proof

$$\mathrm{TRANS}_t := \bigvee_{\langle p,a,b,m,q\rangle \in \Delta} \bigvee_{k \le r(n)} \big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \wedge H_{t+1,k+m} \wedge W_{t+1,k,b} \big] \wedge \dots$$

equivalently:

$$\bigwedge_{k \le r(n)} \bigwedge_{p \in Q} \bigwedge_{a \in \Sigma} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \to \bigvee_{q \in TS(p,a)} S_{t+1,q} \Big]$$

$$\wedge \bigwedge_{k \le r(n)} \bigwedge_{p,q \in Q} \bigwedge_{a \in \Sigma} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \to \bigvee_{m \in TH(p,a,q)} H_{t+1,k+m} \Big]$$

$$\wedge \bigwedge_{k \le r(n)} \bigwedge_{p,q \in Q} \bigwedge_{a \in \Sigma} \bigwedge_{m \in \{-1,0,1\}} \Big[S_{t,p} \wedge H_{t,k} \wedge W_{t,k,a} \wedge S_{t+1,q} \wedge H_{t+1,k+m} \to \bigvee_{b \in TW(p,a,m,q)} W_{t+1,k,b} \Big]$$

$$TW(p,a,m,q) := \{ \, b \in Q \mid \langle p,a,b,m,q\rangle \in \Delta \, \}$$

Proof

Properties of φ_w

- It is in CNF.
- It has length $\sim r(n)^3$.
- It is satisfiable if, and only if, the Turing machine accepts w.

Consequently, the satisfiability problem for PL-formulae in CNF is NP-complete.

Proof

Properties of φ_w

- It is in CNF.
- It has length $\sim r(n)^3$.
- It is satisfiable if, and only if, the Turing machine accepts w.

Consequently, the satisfiability problem for PL-formulae in CNF is NP-complete.

Reduction to 3-CNF

$$\{L_0, L_1, L_2, \dots, L_n\} \quad \mapsto \quad \{L_0, L_1, X\},\ \{\neg X, L_2, \dots, L_n\}$$

(X new variable)

Resolution

Resolution

Resolution Step

The **resolvent** of two clauses

$$C = \{L, A_0, \dots, A_m\} \quad \text{and} \quad C' = \{\neg L, B_0, \dots, B_n\}$$

is the clause

$$\{A_0, \dots, A_m, B_0, \dots, B_n\}.$$

Lemma

Let C be the resolvent of two clauses in Φ. Then

$$\Phi \vDash \Phi \cup \{C\}.$$

Resolution

Resolution Step

The **resolvent** of two clauses

$$C = \{L, A_0, \ldots, A_m\} \quad \text{and} \quad C' = \{\neg L, B_0, \ldots, B_n\}$$

is the clause

$$\{A_0, \ldots, A_m, B_0, \ldots, B_n\}\,.$$

(This is the inverse of the splitting trick from the last slide.)

Lemma

Let C be the resolvent of two clauses in Φ. Then

$$\Phi \vDash \Phi \cup \{C\}\,.$$

The Resolution Method

Observation

If Φ contains the empty clause $\varnothing$, then Φ is not satisfiable.

Resolution Method

Input: a set of clauses Φ

Output: true if Φ is satisfiable, false otherwise.

```
RM(Φ)
    add to Φ all possible resolvents
    repeat until no new clauses are generated
    if ∅ ∈ Φ then
        return false
    else
        return true
```

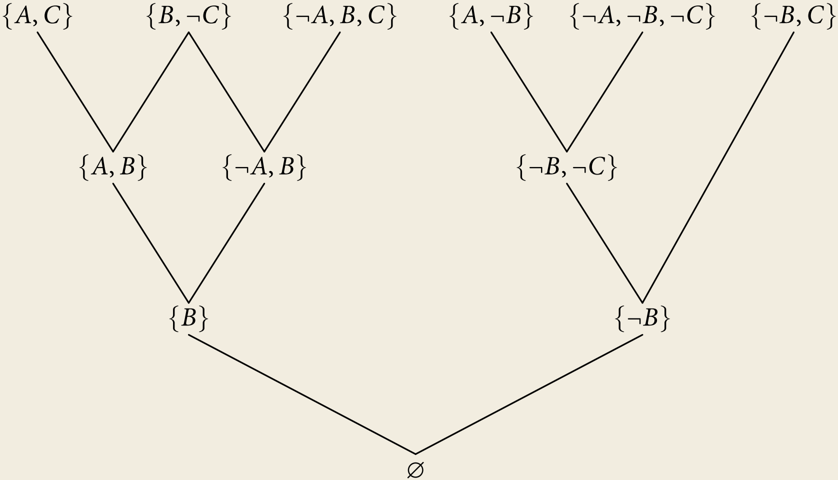
Theorem

The resolution method for propositional logic is **sound** and **complete.**

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

Example



Davis-Putnam Algorithm

Input: a set of clauses Φ
Output: true if Φ is satisfiable, false otherwise.

```
DP(Φ)
    remove all tautological clauses from Φ
    if Φ = ∅ then
        return true
    if Φ = {∅} then
        return false
    select a variable X
    add to Φ all resolvents over X
    remove all clauses containing X or ¬X from Φ
    repeat
```

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

select A: $\{B, C\}$ $\{\neg B, C, \neg C\}$ $\{B, \neg B, C\}$ $\{\neg B, \neg C\}$

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

select A: $\{B, C\}$ $\{\neg B, C, \neg C\}$ $\{B, \neg B, C\}$ $\{\neg B, \neg C\}$

removals: $\{B, \neg C\}$ $\{\neg B, C\}$ $\{B, C\}$ $\{\neg B, \neg C\}$

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

select A: $\{B, C\}$ $\{\neg B, C, \neg C\}$ $\{B, \neg B, C\}$ $\{\neg B, \neg C\}$

removals: $\{B, \neg C\}$ $\{\neg B, C\}$ $\{B, C\}$ $\{\neg B, \neg C\}$

select B: $\{C, \neg C\}$ $\{\neg C\}$ $\{C\}$ $\{C, \neg C\}$

Example

$\{A, C\}$ $\{B, \neg C\}$ $\{\neg A, B, C\}$ $\{A, \neg B\}$ $\{\neg A, \neg B, \neg C\}$ $\{\neg B, C\}$

select A: $\{B, C\}$ $\{\neg B, C, \neg C\}$ $\{B, \neg B, C\}$ $\{\neg B, \neg C\}$

removals: $\{B, \neg C\}$ $\{\neg B, C\}$ $\{B, C\}$ $\{\neg B, \neg C\}$

select B: $\{C, \neg C\}$ $\{\neg C\}$ $\{C\}$ $\{C, \neg C\}$

removals: $\{\neg C\}$ $\{C\}$

Example

$\{A, C\}\ \{B, \neg C\}\ \{\neg A, B, C\}\ \{A, \neg B\}\ \{\neg A, \neg B, \neg C\}\ \{\neg B, C\}$

select A: $\{B, C\}\ \{\neg B, C, \neg C\}\ \{B, \neg B, C\}\ \{\neg B, \neg C\}$

removals: $\{B, \neg C\}\ \{\neg B, C\}\ \{B, C\}\ \{\neg B, \neg C\}$

select B: $\{C, \neg C\}\ \{\neg C\}\ \{C\}\ \{C, \neg C\}$

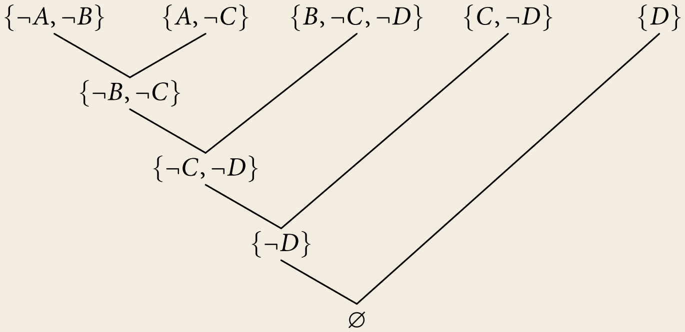
removals: $\{\neg C\}\ \{C\}$

select C: $\varnothing$

Horn formulae

Linear Resolution

A **linear resolution** is a sequence of resolution steps where in each step the resolvent of the previous step is used.



Horn formulae and linear resolution

Horn formulae

A **Horn clause** is a clause C that contains at most one positive literal.

Example

$$A_0 \wedge \cdots \wedge A_n \to B \quad \equiv \quad \{\neg A_0, \ldots, \neg A_n, B\}$$

Horn formulae and linear resolution

Horn formulae

A **Horn clause** is a clause C that contains at most one positive literal.

Example

$$A_0 \wedge \cdots \wedge A_n \to B \quad \equiv \quad \{\neg A_0, \ldots, \neg A_n, B\}$$

Theorem

A set of Horn clauses is unsatisfiable if, and only if, one can use linear resolution to derive the empty clause from it.

SLD Resolution

Linear resolution where the clauses are **sequences** instead of sets and we always resolve the **leftmost literal** of the current clause.

Minimal models

Lemma

Every satisfiable set of Horn-formulae has a minimal model.

Minimal models

Lemma

Every satisfiable set of Horn-formulae has a minimal model.

Algorithm to compute it:

Input: Φ set of Horn-formulae
$T := \varnothing$
repeat
  for all $A_0 \wedge \cdots \wedge A_{n-1} \to B \in \Phi$ **do**
    if $A_0, \ldots, A_{n-1} \in T$ **then**
      $T := T \cup \{B\}$
until T does not change anymore

Minimal models

Lemma

Every satisfiable set of Horn-formulae has a minimal model.

Algorithm to compute it:

Input: Φ set of Horn-formulae
$T := \varnothing$
repeat
  for all $A_0 \wedge \cdots \wedge A_{n-1} \rightarrow B \in \Phi$ **do**
    if $A_0, \ldots, A_{n-1} \in T$ **then**
      $T := T \cup \{B\}$
until T does not change anymore

Theorem

Satisfiability for sets of Horn-formulae can be checked in linear time.

Example

$$B \wedge C \to A \qquad A \wedge D \to B \qquad F \to C \qquad E \to D$$

$$D \wedge E \to A \qquad C \wedge F \to B \qquad 1 \to F$$

Example

$B \wedge C \rightarrow A$ $\quad$ $A \wedge D \rightarrow B$ $\quad$ $\boldsymbol{F} \rightarrow C$ $\quad$ $E \rightarrow D$

$D \wedge E \rightarrow A$ $\quad$ $C \wedge \boldsymbol{F} \rightarrow B$ $\quad$ $1 \rightarrow \boldsymbol{F}$

Example

$$B \wedge C \rightarrow A \qquad A \wedge D \rightarrow B \qquad F \rightarrow C \qquad E \rightarrow D$$

$$D \wedge E \rightarrow A \qquad C \wedge F \rightarrow B \qquad 1 \rightarrow F$$

Example

$$B \wedge C \to A \qquad A \wedge D \to B \qquad F \to C \qquad E \to D$$

$$D \wedge E \to A \qquad C \wedge F \to B \qquad 1 \to F$$

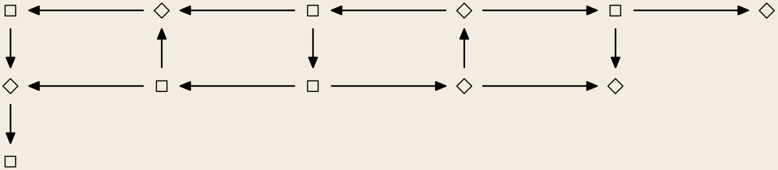
Example

$B \wedge C \to A$ $\quad A \wedge D \to B$ $\quad F \to C$ $\quad E \to D$

$D \wedge E \to A$ $\quad C \wedge F \to B$ $\quad 1 \to F$

Finite Games $\mathcal{G} = \langle V_\diamond, V_\Box, E \rangle$

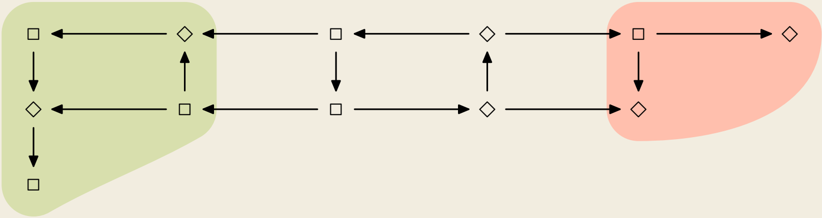
Players $\diamond$ and $\Box$



Winning regions: $W_\diamond, W_\Box$

Finite Games $\mathcal{G} = \langle V_\Diamond, V_\Box, E \rangle$

Players $\Diamond$ and $\Box$



Winning regions: $W_\Diamond, W_\Box$

Reduction

positions

$$V_{\Diamond} = \text{variables } \langle A \rangle \quad \text{and} \quad V_{\Box} = \text{formulae } [A_0 \wedge \cdots \wedge A_{n-1} \to B]$$

edges

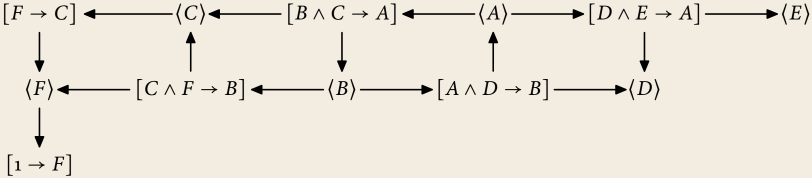
$$\begin{aligned} \langle B \rangle \quad &\to \quad [A_0 \wedge \cdots \wedge A_{n-1} \to B] \\ [A_0 \wedge \cdots \wedge A_{n-1} \to B] \quad &\to \quad \langle A_i \rangle \end{aligned}$$

Lemma

A variable A belongs to $W_{\Diamond}$ iff it is true in the minimal model.

$B \wedge C \to A$ $\quad$ $A \wedge D \to B$ $\quad$ $F \to C$

$D \wedge E \to A$ $\quad$ $C \wedge F \to B$ $\quad$ $1 \to F$



Simple Algorithm

$\mathrm{Win}(v, \sigma)$

if $v \in V_\sigma$ **then**
  if there is an edge $v \to u$ with $\mathrm{Win}(u, \sigma)$ **then**
    return true
  else
    return false

if $v \in V_{\overline{\sigma}}$ **then** (* $\overline{\Diamond} := \Box \quad \overline{\Box} := \Diamond$ *)
  if for every edge $v \to u$ we have $\mathrm{Win}(u, \sigma)$ **then**
    return true
  else
    return false

Linear Algorithm

```
Input: game ⟨V◇, V□, E⟩

forall v ∈ V do
    win[v] := ⊥              (* winner of the position *)
    P[v] := ∅                (* set of predecessors of v *)
    n[v] := 0                (* number of successors of v *)
end

forall ⟨u, v⟩ ∈ E do
    P[v] := P[v] ∪ {u}
    n[u] := n[u] + 1
end

forall v ∈ V◇ do
    if n[v] = 0 then Propagate(v, □)
forall v ∈ V□ do
    if n[v] = 0 then Propagate(v, ◇)
return win
```

```
procedure Propagate(v, σ) =
    if win[v] ≠ ⊥ then return
    win[v] := σ
    forall u ∈ P[v] do
        n[u] := n[u] − 1
        if u ∈ V_σ or n[u] = 0 then Propagate(u, σ)
    end
end
```