Motivation

- Logic is the formal basis of many areas of computer science: digital circuit design, programming language semantics, specification and verification, constraint programming, logic programming, databases, artificial intelligence, knowledge representation, machine learning, ...
- In computational complexity theory: Computational problems from logic are of central importance; they can be used to express computation at various levels.

Syntax

Symbols

The syntax of propositional logic (= Boolean logic) (i.e. the set of well-formed propositional formulae) is based on the following symbols:

- **Boolean variables** (or atoms): \( X = \{x_1, x_2, \ldots\} \).
- **Boolean connectives**: \( \vee, \wedge, \text{ and } \neg \).

Definition

The set of propositional formulae is the smallest set such that

- all Boolean variables are propositional formulae
- if \( \varphi_1 \) and \( \varphi_2 \) are propositional formulae, so are \( \neg \varphi_1, (\varphi_1 \wedge \varphi_2), \) and \( (\varphi_1 \vee \varphi_2) \).

An expression of the form \( x_i \) or \( \neg x_i \) is called a **literal**.
Some notational conventions

- Simplified notation: (((x_1 \lor -x_2) \lor x_3) \lor (x_4 \lor (x_5 \lor -x_6))) is written as x_1 \lor -x_2 \lor x_3 \lor x_4 \lor x_5 \lor -x_6.
- Disjunctions and conjunctions involving n members:
  - \bigvee_{i=1}^{n} \phi_i stands for \phi_1 \lor \cdots \lor \phi_n.
  - \bigwedge_{i=1}^{n} \phi_i stands for \phi_1 \land \cdots \land \phi_n.
- Frequently appearing abbreviations:
  - An implication \phi_1 \rightarrow \phi_2 stands for \neg \phi_1 \lor \phi_2.
  - An equivalence \phi_1 \leftrightarrow \phi_2 stands for (\neg \phi_1 \lor \phi_2) \land (\neg \phi_2 \lor \phi_1).
- The dual (or complement) of a literal \alpha is denoted by \overline{\alpha}, i.e., let \alpha \in X. Then \overline{\alpha} stands for \neg \alpha and \overline{\overline{\alpha}} stands for \alpha.

Satisfaction relation

Definition

Let a truth assignment \( T \) be a mapping from a finite subset \( X' \subset X \) to the set of truth values \{true, false\}. A truth assignment \( T : X' \rightarrow \{true, false\} \) is appropriate to \( \phi \) if \( X(\phi) \subseteq X' \).

Example

Let \( T(x_1) = true, T(x_2) = false \).
Then \( T = x_1 \lor x_2 \), but \( T \neq (x_1 \lor -x_2) \land (-x_1 \land x_2) \).

Semantics

Motivation

How to interpret Boolean expressions?

Observation: Boolean expressions are propositions that are either true or false. They speak about a world where certain atomic propositions (Boolean variables) are either true or false.

Definition

- A truth assignment \( T \) is a mapping from a finite subset \( X' \subset X \) to the set of truth values \{true, false\}.
- Let \( X(\phi) \) be the set of Boolean variables appearing in \( \phi \). A truth assignment \( T : X' \rightarrow \{true, false\} \) is appropriate to \( \phi \) if \( X(\phi) \subseteq X' \).

Logical equivalence

Definition

Expressions \( \phi_1 \) and \( \phi_2 \) are logically equivalent \( (\phi_1 \equiv \phi_2) \) iff for all truth assignments \( T \) appropriate to both of them,
\[ T \models \phi_1 \iff T \models \phi_2. \]

Example

\[
\begin{align*}
(\phi_1 \lor \phi_2) & \equiv (\phi_2 \lor \phi_1) \\
((\phi_1 \land \phi_2) \land \phi_3) & \equiv (\phi_1 \land (\phi_2 \land \phi_3)) \\
\neg \neg \phi & \equiv \phi \\
((\phi_1 \land \phi_2) \lor \phi_3) & \equiv ((\phi_1 \lor \phi_3) \lor (\phi_2 \lor \phi_3)) \\
\neg (\phi_1 \land \phi_2) & \equiv (\neg \phi_1 \lor \neg \phi_2) \\
(\phi_1 \lor \phi_2) & \equiv \phi_1
\end{align*}
\]
Satisfiability and Validity

Definition

- A Boolean expression $\varphi$ is satisfiable iff there is a truth assignment $T$ appropriate to it with $T \models \varphi$.
- A Boolean expression $\varphi$ is valid/a tautology (denoted by $|\models \varphi$) iff for every truth assignment $T$ appropriate to it, $T \models \varphi$.

Proposition

- The following interconnection between satisfiability and validity holds: $\varphi$ is valid $\iff \neg \varphi$ is unsatisfiable.
- Moreover, for any Boolean expressions $\psi_1$ and $\psi_2$, $\psi_1 \equiv \psi_2$ iff $|\models \psi_1 \leftrightarrow \psi_2$ iff $\neg (\psi_1 \leftrightarrow \psi_2)$ is unsatisfiable.

Decision Problems

SAT

INSTANCE: Boolean formula $\varphi$.
QUESTION: Is $\varphi$ satisfiable?

VALIDITY

INSTANCE: Boolean formula $\varphi$.
QUESTION: Is $\varphi$ valid?

Complexity of SAT

Cook-Levin Theorem

SAT is NP-complete.

Proof of the membership

SAT can be decided by the following NP-algorithm:
1. Guess a truth assignment $T$ appropriate to $\varphi$.
2. Check that $T \models \varphi$.

Proof sketch of the hardness (continued)

Let $T$ be a single-string NTM that decides $L$ in $q(|x|)$ for any input $x$ for some polynomial $q(.)$. W.l.o.g., we assume that any computation of $T$ takes exactly $N = q(|x|)$ steps for any input $x$.

Now let $x$ be an arbitrary instance of problem $L$. Then we construct a Boolean formula $R(x)$ over the following propositional atoms:

- $\text{symbol}_{\sigma}^{[\tau, \pi]}$ for $0 \leq \tau \leq N$, $0 \leq \pi \leq N$ and $\sigma \in \Sigma$.
  - Intuitive meaning: at instant $\tau$ of the computation, cell number $\pi$ contains symbol $\sigma$.
- $\text{cursor}^{[\tau, \pi]}$ for $0 \leq \tau \leq N$ and $0 \leq \pi \leq N$.
  - Intuitive meaning: at instant $\tau$, the cursor points to cell number $\pi$.
- $\text{state}_{s}^{[\tau]}$ for $0 \leq \tau \leq N$ and $s \in K$.
  - Intuitive meaning: at instant $\tau$, the NTM $T$ is in state $s$. 

Idea. Let $L$ be an arbitrary problem ($\equiv$ language) in NP. We describe a reduction $R$ which for each string $x$ constructs a Boolean formula $R(x)$ such that $x \in L \iff R(x)$ is satisfiable.
Proof sketch (continued)

The formula \( R(x) \) contains the following groups of conjuncts:

1. **Initialization facts.** Let \( x = x_1 \ldots x_n \).

Then \( R(x) \) contains the following atoms as conjuncts:

\[
\begin{align*}
\text{symbol}_o[0,0] & \quad \text{for } 1 \leq \pi \leq |x|, \text{ where } x_\pi = o \\
\text{symbol}_i[0,\pi] & \quad \text{for } |x| < \pi \leq N \\
\text{cursor}[0,0] & \\
\text{state}_s[0] & 
\end{align*}
\]

2. **Transition rules.** For each pair \((s, \sigma)\) of state \(s\) and symbol \(\sigma\) let \((s, \sigma, s_1', \sigma_1', d_1), \ldots, (s, \sigma, s_k', \sigma_k', d_k)\) denote all possible transitions according to the transition relation \(\Delta\) (for the cursor movements, we write \(d_i \in \{-1, 0, 1\}\) rather than \(d_i \in \{+, -, \rightarrow\}\)).

Then \( R(x) \) contains the following conjuncts for each value of \(\tau\) and \(\pi\) such that \(0 \leq \tau < N \) and \(0 \leq \pi < N \)

\[
\begin{align*}
\text{state}_s[\tau] \land \text{symbol}_o[\tau, \pi] \land \text{cursor}[\tau, \pi] \rightarrow \\
\left(\text{state}_s[\tau + 1, \pi] \land \text{symbol}_o[\tau + 1, \pi] \land \text{cursor}[\tau + 1, \pi + 1] \right) \vee \cdots \vee \\
\left(\text{state}_s[\tau + 1, \pi] \land \text{symbol}_o[\tau + 1, \pi] \land \text{cursor}[\tau + 1, \pi + d_k] \right)
\end{align*}
\]

3. **Uniqueness constraints.** Let \(K = \{s_0, \ldots, s_n\}\) and \(\Sigma = \{\sigma_1, \ldots, \sigma_n\}\).

Then \( R(x) \) contains the following formulae for each value of \(\tau\) and \(\pi\) such that \(0 \leq \tau \leq N, 0 \leq \pi \leq N, 0 \leq i \leq m, \) and \(1 \leq j \leq n\).

\[
\begin{align*}
\text{state}_s[\tau] & \leftrightarrow (\neg\text{state}_s[\tau] \land \cdots \land \neg\text{state}_s[\tau] \land \\
& \land \neg\text{state}_s[\tau + 1] \land \cdots \land \neg\text{state}_s[\tau]) \\
\text{cursor}[\tau, \pi] & \leftrightarrow (\neg\text{cursor}[\tau, 0] \land \cdots \land \neg\text{cursor}[\tau, \pi - 1] \\
& \land \neg\text{cursor}[\tau, \pi + 1] \land \cdots \land \neg\text{cursor}[\tau, N]) \\
\text{symbol}_o[\tau, \pi] & \leftrightarrow (\neg\text{symbol}_o[\tau, \pi] \land \cdots \land \neg\text{symbol}_o[\tau, \pi] \land \\
& \land \neg\text{symbol}_o[\tau, \pi] \land \cdots \land \neg\text{symbol}_o[\tau, \pi])
\end{align*}
\]

5. **Acceptance.** Let \(s_m = \text{"yes"}\).

Then \( R(x) \) contains the following atom as a conjunct:

\[
\text{state}_s[N]
\]

**Complexity of VALIDITY**

**Corollary**

\textbf{VALIDITY} is co-NP-complete.

**Proof**

Recall the following equivalences:

\(\varphi\) is valid \(\iff \neg\varphi\) is unsatisfiable and

\(\varphi\) is unsatisfiable \(\iff \neg\varphi\) is valid.

**Membership.** \textbf{VALIDITY} can be reduced to the co-SAT-problem.

Since SAT is in NP, co-SAT is in co-NP and so is \textbf{VALIDITY}.

**Hardness.** co-SAT can be reduced to the \textbf{VALIDITY}-problem.

Since SAT is NP-hard, co-SAT is co-NP-hard and so is \textbf{VALIDITY}.
Normal Forms

Definition

A formula is in **CNF** (= Conjunctive Normal Form) if it is of the form
\((l_1 \lor \ldots \lor l_n) \land \ldots \land (l_m \lor \ldots \lor l_{mn})\)

A formula is in **DNF** (= Disjunctive Normal Form) if it is of the form
\((l_1 \land \ldots \land l_n) \lor \ldots \lor (l_m \land \ldots \land l_{mn})\)

where each \(l_j\) is a literal (i.e., a Boolean variable or its negation).

Definition

- A disjunction \(l_1 \lor \ldots \lor l_n\) of literals is called a **clause**.
- A conjunction \(l_1 \land \ldots \land l_n\) of literals is called a **implicant**.
- We may assume that normal forms do not have repeated clauses/implicants or repeated literals in clauses/implicants.

Example. \((\neg x_1 \lor \neg x_1 \lor x_2) \equiv (\neg x_1 \lor x_2)\).

CNF/DNF transformation

**Theorem**

Every Boolean expression is equivalent to one in conjunctive (disjunctive) normal form CNF (DNF).

**Proof sketch**

Transformation into a mixed conjunction and disjunction of literals:

- Remove \(\leftrightarrow\) and \(\neg\):  
  \[-\alpha \leftrightarrow \beta \; \Rightarrow \; (\neg \alpha \lor \beta) \land (\neg \beta \lor \alpha) \; (1)\]
  \[-\alpha \rightarrow \beta \; \Rightarrow \; \neg \alpha \lor \beta \; (2)\]

- Push negations in front of Boolean variables:
  \[\neg \neg \alpha \; \Rightarrow \; \alpha \; (3)\]
  \[\neg (\alpha \lor \beta) \; \Rightarrow \; \neg \alpha \land \neg \beta \; (4)\]
  \[\neg (\alpha \land \beta) \; \Rightarrow \; \neg \alpha \lor \neg \beta \; (5)\]

**Example**

Transform \((x_1 \lor x_2) \rightarrow (x_2 \leftrightarrow x_3)\) into CNF.

\[(x_1 \lor x_2) \rightarrow (x_2 \leftrightarrow x_3) \; \Rightarrow (2)\]
\[\neg (x_1 \lor x_2) \lor (x_2 \leftrightarrow x_3) \; \Rightarrow (1)\]
\[\neg (x_1 \lor x_2) \lor ((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (4)\]
\[(\neg x_1 \lor \neg x_2) \lor (((\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \lor (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2))) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (6)\]
\[\neg (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_1 \lor (\neg x_2 \lor x_3)) \land (\neg x_2 \lor (\neg x_2 \lor x_3) \land (\neg x_3 \lor x_2)) \; \Rightarrow (7)\]
SAT and CNF

Theorem
There exists a log-space reduction which reduces any Boolean expression \( \varphi \) into a sat-equivalent Boolean expression \( \psi \) in conjunctive normal form, i.e.: \( \varphi \) is satisfiable \( \iff \psi \) is satisfiable.

Proof sketch

By de Morgan’s laws and the equivalence \( \neg \neg \alpha \equiv \alpha \), negation can be shifted immediately in front of atoms (cf. rewrite rules (4)+(5) resp. (3) above).

The CNF can then be obtained from the resulting formula by successive applications of the following rewrite rule

\[(A \land B) \lor C \land (z \lor A) \land (z \lor B) \land (\neg z \lor C)\]

for some fresh variable \( z \).

Proof sketch (continued)

It can be seen as follows that the above rewrite rule produces a sat-equivalent formula:

- Suppose that \( \Phi = (A \land B) \lor C \) is satisfiable, i.e., there exists a satisfying truth assignment \( I \). We show that \( I \) can be extended to a satisfying truth assignment \( J \) of \( \Psi = (z \lor A) \land (z \lor B) \land (\neg z \lor C) \).
  Case 1: Suppose that \( (A \land B) \) is true in \( I \). We set \( J(z) = \text{false} \).
  Clearly, all conjuncts of \( \Psi \) are true in \( J \).
  Case 2: Suppose that \( C \) is true in \( I \). Then we set \( J(z) = \text{true} \).
  Again, all conjuncts of \( \Psi \) are true in \( J \).

- Suppose that \( \Psi = (z \lor A) \land (z \lor B) \land (\neg z \lor C) \) is satisfiable, i.e., there exists a satisfying truth assignment \( J \) of \( \Psi \). We claim that then \( J \) is also a satisfying truth assignment of \( \Phi = (A \land B) \lor C \).
  Case 1: Suppose that \( J(z) = \text{true} \). By the conjunct \( (z \lor C) \) in \( \Psi \), \( C \) (and thus \( \Phi \)) is true in \( J \).
  Case 2: Suppose that \( J(z) = \text{false} \). By the conjuncts \( (z \lor A) \land (z \lor B) \) in \( \Psi \), both \( A \) and \( B \) (and thus \( \Phi \)) are true in \( J \).
Special cases of SAT

3-SAT

INSTANCE: Boolean formula \( \varphi \) in 3-CNF.
QUESTION: Is \( \varphi \) satisfiable?

2-SAT

INSTANCE: Boolean formula \( \varphi \) in 2-CNF (i.e., each clause consists of exactly 2 literals).
QUESTION: Is \( \varphi \) satisfiable?

HORNSAT

INSTANCE: Boolean formula \( \varphi \) in CNF, s.t. each clause is in Horn form (i.e., each clause contains at most one positive literal).
QUESTION: Is \( \varphi \) satisfiable?

3-SAT

Theorem

3-SAT is \( \text{NP} \)-complete.

Proof

- Membership is clear since 3-SAT is a special case of SAT.
- Hardness follows from the NP-hardness of SAT and from the fact that any Boolean expression \( \varphi \) can be reduced into a sat-equivalent Boolean expression \( \psi \) in 3-CNF (i.e.: \( \psi \) is in CNF and every clause of \( \psi \) consists of exactly 3 literals).

Corollary

The VALIDITY-problem remains co-NP-complete even if the formulae are restricted to 3-DNF (i.e., DNF where each implicant consists of exactly 3 literals).

Proof sketch of the membership (continued)

Idea of the SAT-test.

Compute the set \( Y \) of all variables that are logically implied by the facts and rules in \( \varphi \).

- If there exists a goal \( \neg q_1 \lor \cdots \lor \neg q_n \) in \( \varphi \), s.t. every \( q_i \) is in \( Y \), then \( \varphi \) is unsatisfiable.
- Otherwise \( \varphi \) is satisfiable. Indeed, we get a model of \( \varphi \) by setting all propositional variables in \( Y \) to true and all other variables to false.
Complexity Theory 4. Boolean Logic 4.7. 2-SAT

2-SAT

Theorem

2-SAT is co-NL-complete.

Proof sketch of the membership

Recall the 2-SAT-algorithm from the “Formale Methoden” lecture. Given an arbitrary instance \( \varphi \) of 2-SAT, we define the graph \( G(\varphi) \) as follows:

- The variables of \( \varphi \) and their negations form the vertices of \( G(\varphi) \).
- There is an arc \((\alpha, \beta)\) iff there is a clause \( \overline{\alpha} \lor \beta \) or \( \beta \lor \overline{\alpha} \) in \( \varphi \), where \( \overline{\alpha} \) is the complement of \( \alpha \), i.e.: If \( \alpha \) is true in some satisfying assignment \( I \) of \( \varphi \), then \( \beta \) must also be true in \( I \).
- It can be shown that \( \varphi \) is unsatisfiable iff there is a variable \( x \) such that there are paths from \( x \) to \( \neg x \) and from \( \neg x \) to \( x \) in \( G(\varphi) \).

NL-algorithm for the unsatisfiability of \( \varphi \). Guess a variable \( x \) and check that \( x \) is reachable from \( \neg x \) and \( \neg x \) is reachable from \( x \) in \( G(\varphi) \).

Proof sketch of the hardness

3. Uniqueness constraints. It suffices to express the condition that, at any time instant \( \tau \), the machine is in at most one state, the cursor is in at most one position, and each tape cell contains at most one symbol, e.g.: for all \( i \neq j, \pi \neq \pi' \), \( R(x) \) contains conjuncts of the following form:

\[
\neg \text{state}_{s}[\tau] \lor \neg \text{state}_{\pi}[\tau] \\
\neg \text{cursor}[\tau, \pi] \lor \neg \text{cursor}[\tau, \pi'] \\
\neg \text{symbol}_{s}[r, \pi] \lor \neg \text{symbol}_{\pi}[r, \pi] \\
\text{symbol}_{s}[r, \pi] \lor \text{symbol}_{\pi}[r, \pi']
\]

4. Inertia rules. No changes required. Conjuncts in \( R(x) \):

\[
\text{symbol}_{s}[r, \pi], \text{cursor}[\tau, \pi'] \rightarrow \text{symbol}_{s}[r + 1, \pi] \\
\text{symbol}_{s}[r, \pi'], \text{cursor}[\tau, \pi] \rightarrow \text{symbol}_{s}[r + 1, \pi']
\]

5. Acceptance. No changes required. Conjunct in \( R(x) \):

\[
\text{state}_{s}[N], \text{where } s_{m} = \text{"yes"}
\]

Example

\[
\psi = (x_1 \lor x_2) \land (x_1 \lor \neg x_3) \land (\neg x_1 \lor x_2) \land (x_2 \lor x_3)
\]
\[ \psi = (x_1 \lor x_2) \land (x_1 \lor \neg x_3) \land (\neg x_1 \lor x_2) \land (x_2 \lor x_3) \]

\[ \neg x_1 \Rightarrow x_2 \]
\[ \neg x_2 \Rightarrow \neg x_1 \]

\[ \psi = (x_1 \lor x_2) \land (x_1 \lor \neg x_3) \land (\neg x_1 \lor x_2) \land (x_2 \lor x_3) \]

\[ \neg x_2 \Rightarrow x_3 \]
\[ \neg x_3 \Rightarrow x_2 \]
Proof sketch of the hardness

We reduce \textsc{Reachability} to the \textsc{co-2-SAT}-problem. Let \((G, s, t)\) be an arbitrary instance of \textsc{Reachability}, where \(G = (V, E)\) is a graph and \(s, t\) are nodes in \(V\). W.l.o.g., we may assume that \(G\) contains no isolated nodes (i.e., every node in \(V\) is adjacent to at least one edge).

We construct the following instance \(R(G, s, t) = \varphi\) of \textsc{co-2-SAT}:

- The set of variables in \(\varphi\) is \(V\).
- For every edge \((a, b)\) in \(G\), the formula \(\varphi\) contains the clause \(\neg a \vee b\) (or, equivalently \(a \rightarrow b\)).
- Finally, the formula \(\varphi\) also contains the unit clauses \(s\) and \(\neg t\).

Clearly, this reduction is feasible in log-space. It remains to prove its correctness.

---

**Remarks**

- By the Immerman-Szelepcényi Theorem we know that \(\text{co-NL} = \text{NL}\).
- Hence, the co-NL-completeness of \textsc{2-SAT} immediately implies that \textsc{2-SAT} is also NL-complete.
- Three of the most fundamental complexity classes (NL, P, NP) have thus been shown to contain natural variants of \textsc{SAT} as complete problems, namely \textsc{2-SAT}, \textsc{HornSAT}, and \textsc{3-SAT}, respectively.

---

**Efficient Solution of \textsc{HornSAT}**

**Basic Step**

INPUT: a set \(X\) of variables, set \(\Pi\) of rules (a prop. logic program)
OUTPUT: Compute the least fixed-point (denoted as \(X^+\)) of the immediate consequence operator w.r.t. rules \(\Pi\) applied to \(X\).

**Remarks**

- \(X^+\) contains all variables derivable by \(\Pi\) from \(X\).
- Clearly, for every variable \(z\), we have \(z \in X^+\) iff \(X \cup \Pi \models z\).

**Motivation**

The basic step occurs (in different terminology) in several areas of computer science, like deriving functional dependencies in a relational schema, graph reachability, reachability in a \(CFG\), etc.
Computing the least fixed-point $X^+$

**Proposition**

Let $\Pi$ be a set of rules over variables $V$ and let $X \subseteq V$. The least fixed-point $X^+$ of $X$ can be computed in polynomial time.

**Proof**

A straightforward polynomial-time algorithm works as follows:

1. $Y := X$;
2. while $\exists r \in \Pi$, s.t. $\text{body}(r) \subseteq Y$ and $\text{head}(r) \notin Y$ do
   1. $Y := Y \cup \{\text{head}(r)\}$;
3. endwhile;
4. return $Y$;

Algorithm of Beeri and Bernstein

**Data structures**

Input: Set of rules $\Pi$ over $V$, variable set $X \subseteq V$
Output: Least fixed point $X^+$ w.r.t. immediate consequence operator
Auxiliary data structures:
- count: array of integers, index: each $r \in \Pi$, $L$: array of lists of rules, index: each $z \in V$

**Initialization**

unmark all members of $V$;
for each $z \in V$ do $L(z) := \text{empty-list}$;
for each $r \in \Pi$ do $\text{count}(r) := |\text{body}(r)|$;
for each $z \in \text{body}(r)$ do add $r$ to the list $L(z)$;
end for;

Complexity of Computing $X^+$

**Motivation**

Complexity of computing $X^+$ for some subset $X \subseteq V$

- Straightforward algorithm: works in time $O(|V|^2 \cdot |\Pi|)$.
  - number of iterations: $|V|$
  - in each iteration, scan through all rules $r \in \Pi$ once
    $\Rightarrow |\Pi|$ upper bound.
  - Check if $\text{body}(r) \subseteq Y$ holds $\Rightarrow |V|$ upper bound.
- In (Beeri/Bernstein, 1979), it was shown (for the corresponding problem on functional dependencies) that $X^+$ can be computed in linear time. More precisely, the algorithm works in time $O(|V| + \|\Pi\|)$.

Algorithm of Beeri and Bernstein (continued)

**Computation of $X^+$**

1. $Y := X$;
2. while $Y$ contains an unmarked element $z$ do
   1. mark $z$;
      1. for each $r \in L(z)$ do
         1. $\text{count}(r) := \text{count}(r) - 1$;
         2. if $\text{count}(r) = 0$ then $Y := Y \cup \{\text{head}(r)\}$;
      end for;
   end for;
3. endwhile;
4. return $Y$;
Complexity Theory

4. Boolean Logic

4.8. Efficient Solution of HORNSAT

Algorithm of Beeri and Bernstein (continued)

Correctness of the algorithm (rough proof sketch)

The proof goes via the following loop invariant of the while-loop:

- All marked elements are in \( Y \),
- \( Y \subseteq X^+ \),
- For all \( r \in \Pi \), we have count\( (r) = |\{ z \in \text{body}(r) \mid z \text{ is not marked}\}| \).

Upper bound \( O(|V| + ||\Pi||) \) on time complexity

- Initialization: takes time \( O(|V| + ||\Pi||) \)
- while-loop:
  - Each element is marked at most once.
  - Altogether, the counts are decremented at most \( ||\Pi|| \) times.
  - The innermost loop goes through all the heads of rules once. Hence, once more \( O(||\Pi||) \) is needed.

Decision Procedure for HORNSAT

INSTANCE: Boolean formula \( \varphi \) in CNF over the propositional variables \( V \), s.t. each clause is in Horn form (i.e., has at most one positive literal).

QUESTION: Is \( \varphi \) satisfiable?

ALL these problems can be solved in linear time.

Learning Objectives

- Recapitulation of the syntax and semantics of Boolean expressions (= propositional formulae).
- Satisfiability and validity of Boolean expressions.
- Normal forms of Boolean expressions: CNF, DNF, 3-CNF, 2-CNF.
- Difference between equivalence and sat-equivalence.
- Two fundamental NP-complete decision problems: SAT and 3-SAT.
- Two tractable special cases of SAT: HORNSAT and 2-SAT.
- Linear-time algorithm for HORNSAT and related problems.