Logic and Automated Reasoning

Andrei Voronkov
(University of Manchester)

Outline

Computer Systems and Correctness

Suppose we design a (complex) computer system, which may contain various components, for example, hardware, software etc.
We have high requirements to the correctness of the system (safety, reliability, security, consistent state, no deadlocks etc.)
How can one achive a 100% safety?
Computer systems are becoming increasingly unreliable.

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?
We can write a Spec#-specification.
How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?

We can write a Spec#-specification.

How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?
We can write a Spec#-specification.
How can we **prove** automatically that the program is correct w.r.t. this specification?

Small Example: Software

Consider the following fragment of a C++ program:

```
int sumOfFirstNIntegers(int n)
  requires n >= 0
  ensures result = n * (n+1) / 2
{
  int sum = 0;
  for (i = n;i != 0;i = i-1) { sum = sum+i; }
  return sum;
}
```

We know that

$$1 + \ldots + n = \frac{n \cdot (n+1)}{2}$$

Is it true that for all integer n the program returns $\frac{n \cdot (n+1)}{2}$?
We can write a Spec#-specification.
How can we **prove** automatically that the program is correct w.r.t. this specification?

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.

We want to be sure that C_2 is correct, that is, it will behave according to some specification.

If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.
We want to be sure that C_2 is correct, that is, it will behave according to some specification.
If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Another example: circuit design

We used a circuit C_1 in a processor and would like to replace it by another circuit C_2. For example, we may believe that the use of C_2 results in a lower energy consumption.
We want to be sure that C_2 is correct, that is, it will behave according to some specification.
If we know that C_1 is correct, it is sufficient to **prove** that C_2 is functionally equivalent to C_1.

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Automated Theorem Proving. Example

Group theory theorem: if a group satisfies the identity $x^2 = 1$, then it is commutative.

More formally: in a group "assuming that $x^2 = 1$ for all x prove that $x \cdot y = y \cdot x$ holds for all x, y."

What is implicit: axioms of the group theory.

$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Formulation in First-Order Logic

Axioms (of group theory):
$$\forall x(1 \cdot x = x)$$
$$\forall x(x^{-1} \cdot x = 1)$$
$$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$$

Assumptions:
$$\forall x(x \cdot x = 1)$$

Conjecture:
$$\forall x \forall y(x \cdot y = y \cdot x)$$

Formulation in First-Order Logic

Axioms (of group theory):	$\forall x(1 \cdot x = x)$
	$\forall x(x^{-1} \cdot x = 1)$
	$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$
Assumptions:	$\forall x(x \cdot x = 1)$
Conjecture:	$\forall x \forall y(x \cdot y = y \cdot x)$

Formulation in First-Order Logic

Axioms (of group theory):	$\forall x(1 \cdot x = x)$
	$\forall x(x^{-1} \cdot x = 1)$
	$\forall x \forall y \forall z((x \cdot y) \cdot z = x \cdot (y \cdot z))$
Assumptions:	$\forall x(x \cdot x = 1)$
Conjecture:	$\forall x \forall y(x \cdot y = y \cdot x)$

In the TPTP Syntax

TPTP library (Thousands of Problems for Theorem Provers), www.tptp.org.

```
%---- 1 * x = 1
fof(left_identity,axiom,
  mult(e,X) = X.
%---- i(x) * x = 1
fof(left_inverse,axiom,
  mult(inverse(X),X) = e).
%---- (x * y) * z = x * (y * z)
fof(associativity,axiom,
  mult(mult(X,Y),Z) = mult(X,mult(Y,Z))).
%---- x * x = 1
fof(group_of_order_2,hypothesis,
  mult(X,X) = e).
%---- prove x * y = y * x
fof(commutativity,conjecture,
  mult(X,Y) = mult(Y,X)).
```

Example: Proof by Vampire

.....

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.

Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Theorem Provers

Theorem Prover: a system that can prove theorems automatically.
Two kinds of provers:

- automatic provers;
- interactive provers, or proof assistants.

Logics:

- in automatic provers mainly first-order logic (with built-in equality);
- in interactive provers higher-order logics or type theories.

This course will be mainly about fully automatic theorem provers for first-order logic.

Main applications

- Software and hardware verification;
- Static analysis of programs;
- Query answering in first-order knowledge bases (ontologies), Semantic Web;
- Theorem proving in mathematics, especially in algebra;
- Verification of cryptographic protocols;
- Circuit design;
- Constraint satisfaction;
- Planning;
- Databases (semantics and query optimisation);
- Solving exercises for this course ◡̈

What We Expect of an Automatic Theorem Prover

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

What We Expect of an Automatic Theorem Prover

Input:

- a set of axioms (first order formulas) or clauses;
- a conjecture (first-order formula or set of clauses).

Output:

- proof (hopefully).

Outline

Propositional logic: syntax

Assume a countable set of boolean variables.
Propositional formula:

- Every boolean variable is a formula, also called atomic formula, or simply atom.
- $\top$ and $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \wedge \ldots \wedge A_n)$ and $(A_1 \vee \ldots \vee A_n)$ are formulas.
- If A is a formula, then $\neg A$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

The symbols $\top, \bot, \wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called logical connectives.

Propositional logic: syntax

Assume a countable set of boolean variables.
Propositional formula:

- Every boolean variable is a formula, also called atomic formula, or simply atom.
- $\top$ and $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \wedge \ldots \wedge A_n)$ and $(A_1 \vee \ldots \vee A_n)$ are formulas.
- If A is a formula, then $\neg A$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

The symbols $\top, \bot, \wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called connectives.

Propositional logic: syntax

Assume a countable set of boolean variables.
Propositional formula:

- Every boolean variable is a formula, also called atomic formula, or simply atom.
- $\top$ and $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \wedge \ldots \wedge A_n)$ and $(A_1 \vee \ldots \vee A_n)$ are formulas.
- If A is a formula, then $\neg A$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

The symbols $\top, \bot, \wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called connectives.

Propositional logic: syntax

Assume a countable set of boolean variables.
Propositional formula:

- Every boolean variable is a formula, also called atomic formula, or simply atom.
- $\top$ and $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \wedge \ldots \wedge A_n)$ and $(A_1 \vee \ldots \vee A_n)$ are formulas.
- If A is a formula, then $\neg A$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.

The symbols $\top, \bot, \wedge, \vee, \neg, \rightarrow, \leftrightarrow$ are called connectives.

Connectives

Connective	Name	Priority
$\top$	verum	
$\bot$	falsum	
$\neg$	negation	4
$\wedge$	conjunction	3
$\vee$	disjunction	3
$\longrightarrow$	implication	2
$\longleftrightarrow$	equivalence	1

Parsing Formulas

We normally omit parenthesis in mathematical expressions and use priorities to disambiguate them.

For example, in arithmetic we know that the expression

$$x \cdot y + 2 \cdot z$$

is equivalent to

$$(x \cdot y) + (2 \cdot z),$$

since $\cdot$ has a higher priority than $+$.

We will also use priorities to disambiguate formulas.

Parsing Formulas

We normally omit parenthesis in mathematical expressions and use priorities to disambiguate them.

For example, in arithmetic we know that the expression

$$x \cdot y + 2 \cdot z$$

is equivalent to

$$(x \cdot y) + (2 \cdot z),$$

since $\cdot$ has a higher priority than $+$.

We will also use priorities to disambiguate formulas.

Parsing Formulas

We normally omit parenthesis in mathematical expressions and use priorities to disambiguate them.

For example, in arithmetic we know that the expression

$$x \cdot y + 2 \cdot z$$

is equivalent to

$$(x \cdot y) + (2 \cdot z),$$

since $\cdot$ has a higher priority than $+$.

We will also use priorities to disambiguate formulas.

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$((\neg A) \wedge B) \rightarrow (C \vee D) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Parsing: Example

Let's parse $\neg A \wedge B \rightarrow C \vee D \leftrightarrow E$.

Connective	Priority
$\top$	
$\bot$	
$\neg$	4
$\wedge$	3
$\vee$	3
$\rightarrow$	2
$\leftrightarrow$	1

Inside-out (starting with the highest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Outside-in (starting with the lowest priority connectives):

$$(((\neg A) \wedge B) \rightarrow (C \vee D)) \leftrightarrow E.$$

Semantics, Interpretation

Consider an arithmetical expression, for example

$$x \cdot y + 2 \cdot z.$$

In arithmetic the meaning of expressions with variables is defined as follows.

Take a mapping from variables (integer) values, for example

$$\{x \mapsto 1, y \mapsto 7, z \mapsto -3\}.$$

Then, under this mapping the expression has the value 1. In other words, when we interpret variables as values, we can compute the value of the expression.

Semantics, Interpretation

Consider an arithmetical expression, for example

$$x \cdot y + 2 \cdot z.$$

In arithmetic the meaning of expressions with variables is defined as follows.
Take a mapping from variables (integer) values, for example

$$\{x \mapsto 1, y \mapsto 7, z \mapsto -3\}.$$

Then, under this mapping the expression has the value 1. In other words, when we interpret variables as values, we can compute the value of the expression.

Semantics, Interpretation

Consider an arithmetical expression, for example

$$x \cdot y + 2 \cdot z.$$

In arithmetic the meaning of expressions with variables is defined as follows.
Take a mapping from variables (integer) values, for example

$$\{x \mapsto 1, y \mapsto 7, z \mapsto -3\}.$$

Then, under this mapping the expression has the value 1. In other words, when we interpret variables as values, we can compute the value of the expression.

Semantics, Interpretation

Likewise, the semantics of propositional formulas can be defined by assigning boolean values to variables.

- A boolean value, also called a truth value, is either true (denoted 1) or false (denoted 0).
- An interpretation for a set P of boolean variables is a mapping $I : P \to \{1, 0\}$.
- Interpretations are also called truth assignments.

Semantics, Interpretation

Likewise, the semantics of propositional formulas can be defined by assigning boolean values to variables.

- A boolean value, also called a truth value, is either true (denoted 1) or false (denoted 0).
- An interpretation for a set P of boolean variables is a mapping $I : P \to \{1, 0\}$.
- Interpretations are also called truth assignments.

Semantics, Interpretation

Likewise, the semantics of propositional formulas can be defined by assigning boolean values to variables.

- A boolean value, also called a truth value, is either true (denoted 1) or false (denoted 0).
- An interpretation for a set P of boolean variables is a mapping $I : P \to \{1, 0\}$.
- Interpretations are also called truth assignments.

Semantics, Interpretation

Likewise, the semantics of propositional formulas can be defined by assigning boolean values to variables.

- A boolean value, also called a truth value, is either true (denoted 1) or false (denoted 0).
- An interpretation for a set P of boolean variables is a mapping $I : P \to \{1, 0\}$.
- Interpretations are also called truth assignments.

Interpreting formulas

Extend I to all formulas:

1. $I(\top) = 1$ and $I(\bot) = 0$.
2. $I(A_1 \wedge \ldots \wedge A_n) = 1$ if and only if $I(A_i) = 1$ for all i.
3. $I(A_1 \vee \ldots \vee A_n) = 1$ if and only if $I(A_i) = 1$ for some i.
4. $I(\neg A) = 1$ if and only if $I(A) = 0$.
5. $I(A_1 \rightarrow A_2) = 1$ if and only if $I(A_1) = 0$ or $I(A_2) = 1$.
6. $I(A_1 \leftrightarrow A_2) = 1$ if and only if $I(A_1) = I(A_2)$.

Operation tables

$I(A_1 \vee A_2) = 1$ if and only if $I(A_1) = 1$ or $I(A_2) = 1$.
$I(A_1 \leftrightarrow A_2) = 1$ if and only if $I(A_1) = I(B_2)$.

$\wedge$	1	0
1	1	0
0	0	0

$\vee$	1	0
1	1	1
0	1	0

$\neg$	
1	0
0	1

$\rightarrow$	1	0
1	1	0
0	1	1

$\leftrightarrow$	1	0
1	1	0
0	0	1

Therefore, every connective can be considered as a function on truth values.

Operation tables

$I(A_1 \vee A_2) = 1$ if and only if $I(A_1) = 1$ or $I(A_2) = 1$.

$I(A_1 \leftrightarrow A_2) = 1$ if and only if $I(A_1) = I(B_2)$.

$\wedge$	1	0
1	1	0
0	0	0

$\vee$	1	0
1	1	1
0	1	0

$\neg$	
1	0
0	1

$\rightarrow$	1	0
1	1	0
0	1	1

$\leftrightarrow$	1	0
1	1	0
0	0	1

Therefore, every connective can be considered as a function on truth values.

Operation tables

$I(A_1 \vee A_2) = 1$ if and only if $I(A_1) = 1$ or $I(A_2) = 1$.
$I(A_1 \leftrightarrow A_2) = 1$ if and only if $I(A_1) = I(B_2)$.

$\wedge$	1	0
1	1	0
0	0	0

$\vee$	1	0
1	1	1
0	1	0

$\neg$	
1	0
0	1

$\rightarrow$	1	0
1	1	0
0	1	1

$\leftrightarrow$	1	0
1	1	0
0	0	1

Therefore, every connective can be considered as a function on truth values.

Operation tables

$I(A_1 \vee A_2) = 1$ if and only if $I(A_1) = 1$ or $I(A_2) = 1$.
$I(A_1 \leftrightarrow A_2) = 1$ if and only if $I(A_1) = I(B_2)$.

$\wedge$	1	0
1	1	0
0	0	0

$\vee$	1	0
1	1	1
0	1	0

$\neg$	
1	0
0	1

$\rightarrow$	1	0
1	1	0
0	1	1

$\leftrightarrow$	1	0
1	1	0
0	0	1

Therefore, every connective can be considered as a function on truth values.

Satisfiability, validity

- If $I(A) = 1$, then we say that the formula A is true in I and that I satisfies A and that I is a model of A, denoted by $I \models A$.
- If $I(A) = 0$, then we say that the formula A is false in I.
- A is satisfiable (valid) if it is true in some (every) interpretation.
- Two formulas A and B are called equivalent, denoted by $A \equiv B$ if they have the same models.

Satisfiability, validity

- If $I(A) = 1$, then we say that the formula A is true in I and that I satisfies A and that I is a model of A, denoted by $I \models A$.
- If $I(A) = 0$, then we say that the formula A is false in I.
- A is satisfiable (valid) if it is true in some (every) interpretation.
- Two formulas A and B are called equivalent, denoted by $A \equiv B$ if they have the same models.

Satisfiability, validity

- If $I(A) = 1$, then we say that the formula A is true in I and that I satisfies A and that I is a model of A, denoted by $I \models A$.
- If $I(A) = 0$, then we say that the formula A is false in I.
- A is satisfiable (valid) if it is true in some (every) interpretation.
- Two formulas A and B are called equivalent, denoted by $A \equiv B$ if they have the same models.

Satisfiability, validity

- If $I(A) = 1$, then we say that the formula A is true in I and that I satisfies A and that I is a model of A, denoted by $I \models A$.
- If $I(A) = 0$, then we say that the formula A is false in I.
- A is satisfiable (valid) if it is true in some (every) interpretation.
- Two formulas A and B are called equivalent, denoted by $A \equiv B$ if they have the same models.

Examples

$A \rightarrow A$ and $A \vee \neg A$ are valid for all formulas A.

Evidently, every valid formula is also satisfiable.

$A \wedge \neg A$ is unsatisfiable.

Formula p, where p is a boolean variable, is satisfiable but not valid.

Examples

$A \rightarrow A$ and $A \vee \neg A$ are valid for all formulas A.

Evidently, every valid formula is also satisfiable.

$A \wedge \neg A$ is unsatisfiable.

Formula p, where p is a boolean variable, is satisfiable but not valid.

Examples

$A \rightarrow A$ and $A \vee \neg A$ are valid for all formulas A.

Evidently, every valid formula is also satisfiable.

$A \wedge \neg A$ is unsatisfiable.

Formula p, where p is a boolean variable, is satisfiable but not valid.

Examples

$A \rightarrow A$ and $A \vee \neg A$ are valid for all formulas A.

Evidently, every valid formula is also satisfiable.

$A \wedge \neg A$ is unsatisfiable.

Formula p, where p is a boolean variable, is satisfiable but not valid.

Examples: equivalences

For all formulas A and B, the following equivalences hold.

$$A \to \bot \equiv \neg A; \tag{1}$$
$$\top \to A \equiv A; \tag{2}$$
$$A \to B \equiv \neg(A \wedge \neg B); \tag{3}$$
$$A \wedge B \equiv \neg(\neg A \vee \neg B); \tag{4}$$
$$A \vee B \equiv \neg A \to B. \tag{5}$$

Connections between these notions

1. A formula A is valid if and only if $\neg A$ is unsatisfiable.
2. A formula A is satisfiable if and only if $\neg A$ is not valid.
3. A formula A is valid if and only if A is equivalent to $\top$.
4. Formulas A and B are equivalent if and only if the formula $A \leftrightarrow B$ is valid.

Connections between these notions

1. A formula A is valid if and only if $\neg A$ is unsatisfiable.
2. A formula A is satisfiable if and only if $\neg A$ is not valid.
3. A formula A is valid if and only if A is equivalent to $\top$.
4. Formulas A and B are equivalent if and only if the formula $A \leftrightarrow B$ is valid.

Equivalent replacement

We denote by $A[B]$ a formula A with a fixed occurrence of a subformula B. If we use this notation we can also write $A[B']$ to denote the formula obtained from A by replacing this occurrence of B by B'.

Lemma (Equivalent Replacement)

Let I be an interpretation and $I \models A_1 \leftrightarrow A_2$. Then $I \models B[A_1] \leftrightarrow B[A_2]$.

Theorem (Equivalent Replacement)

Let $A_1 \equiv A_2$. Then Then $B[A_1] \equiv B[A_2]$.

Equivalent replacement

We denote by $A[B]$ a formula A with a fixed occurrence of a subformula B. If we use this notation we can also write $A[B']$ to denote the formula obtained from A by replacing this occurrence of B by B'.

Lemma (Equivalent Replacement)

Let I be an interpretation and $I \models A_1 \leftrightarrow A_2$. Then $I \models B[A_1] \leftrightarrow B[A_2]$.

Theorem (Equivalent Replacement)

Let $A_1 \equiv A_2$. Then Then $B[A_1] \equiv B[A_2]$.

Equivalent replacement

We denote by $A[B]$ a formula A with a fixed occurrence of a subformula B. If we use this notation we can also write $A[B']$ to denote the formula obtained from A by replacing this occurrence of B by B'.

Lemma (Equivalent Replacement)

Let I be an interpretation and $I \models A_1 \leftrightarrow A_2$. Then $I \models B[A_1] \leftrightarrow B[A_2]$.

Theorem (Equivalent Replacement)

Let $A_1 \equiv A_2$. Then Then $B[A_1] \equiv B[A_2]$.

Propositional Satisfiability Problem

Given a propositional formula A, check wheter it is satisfiable or not.

Desirable: if A is satisfiable, try to find a satisfying assignment for A, that is, a model of A.

Propositional Satisfiability Problem

Given a propositional formula A, check wheter it is satisfiable or not.

Desirable: if A is satisfiable, try to find a satisfying assignment for A, that is, a model of A.

Russian spy puzzle

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans. Moreover, every Russian must be a spy.

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”. It is known that Stirlitz always tells the truth when he is joking.

We have to establish that Eismann is not a Russian spy.

How can we solve problems of this kind?

Russian spy puzzle

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans. Moreover, every Russian must be a spy.

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”. It is known that Stirlitz always tells the truth when he is joking.

We have to establish that Eismann is not a Russian spy.

How can we solve problems of this kind?

Russian spy puzzle

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans. Moreover, every Russian must be a spy.

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”. It is known that Stirlitz always tells the truth when he is joking.

We have to establish that Eismann is not a Russian spy.

How can we solve problems of this kind?

Russian spy puzzle

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans. Moreover, every Russian must be a spy.

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”. It is known that Stirlitz always tells the truth when he is joking.

We have to establish that Eismann is not a Russian spy.

How can we solve problems of this kind?

Formalisation in propositional logic

Introduce propositional variables XY with the following meaning in mind:

$X \in \{R, G, S\}$ (denoting Russian, German, Spy)
$Y \in \{S, M, E\}$ (denoting Stirlitz, Müller, Eismann)

For example,

SE : Eismann is a Spy
RS : Stirlitz is Russian

Formalisation in propositional logic

Introduce propositional variables XY with the following meaning in mind:

$X \in \{R, G, S\}$ (denoting Russian, German, Spy)
$Y \in \{S, M, E\}$ (denoting Stirlitz, Müller, Eismann)

For example,

SE : Eismann is a Spy
RS : Stirlitz is Russian

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”.

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”.

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: "you know, Müller, you are as German as I am Russian".

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: "you know, Müller, you are as German as I am Russian".

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: "you know, Müller, you are as German as I am Russian".

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Formalisation in propositional logic

There are three persons: Stirlitz, Müller, and Eismann. It is known that exactly one of them is Russian, while the other two are Germans.

$$(RS \wedge GM \wedge GE) \vee (GS \wedge RM \wedge GE) \vee (GS \wedge GM \wedge RE).$$

Moreover, every Russian must be a spy.

$$(RS \rightarrow SS) \wedge (RM \rightarrow SM) \wedge (RE \rightarrow SE).$$

When Stirlitz meets Müller in a corridor, he makes the following joke: “you know, Müller, you are as German as I am Russian”.

$$RS \leftrightarrow GM.$$

We have to establish that Eismann is not a Russian spy.

$$\neg(RE \wedge SE).$$

Hidden: Russians are not Germans.

$$(RS \leftrightarrow \neg GS) \wedge (RM \leftrightarrow \neg GM) \wedge (RE \leftrightarrow \neg GE).$$

Why satisfiability?

A formula A is a logical consequence of formulas $A_1, \ldots, A_n$, or follows from $A_1, \ldots, A_n$, if every model of $A_1, \ldots, A_n$ is also a model of A.

Note that A is not a logical consequence of $A_1, \ldots, A_n$ if and only if the set of formulas $A_1, \ldots, A_n, \neg A$ is satisfiable.

We have to determine whether the fact that Eismann is not a Russian spy follows from the conditions of the puzzle.

Therefore, the problem of solving the puzzle is an instance of the satisfaibility problem.

Why satisfiability?

A formula A is a logical consequence of formulas $A_1, \ldots, A_n$, or follows from $A_1, \ldots, A_n$, if every model of $A_1, \ldots, A_n$ is also a model of A.

Note that A is not a logical consequence of $A_1, \ldots, A_n$ if and only if the set of formulas $A_1, \ldots, A_n, \neg A$ is satisfiable.

We have to determine whether the fact that Eismann is not a Russian spy follows from the conditions of the puzzle.

Therefore, the problem of solving the puzzle is an instance of the satisfaibility problem.

Why satisfiability?

A formula A is a logical consequence of formulas $A_1, \ldots, A_n$, or follows from $A_1, \ldots, A_n$, if every model of $A_1, \ldots, A_n$ is also a model of A.

Note that A is not a logical consequence of $A_1, \ldots, A_n$ if and only if the set of formulas $A_1, \ldots, A_n, \neg A$ is satisfiable.

We have to determine whether the fact that Eismann is not a Russian spy follows from the conditions of the puzzle.

Therefore, the problem of solving the puzzle is an instance of the satisfaibility problem.

Why satisfiability?

A formula A is a logical consequence of formulas $A_1, \ldots, A_n$, or follows from $A_1, \ldots, A_n$, if every model of $A_1, \ldots, A_n$ is also a model of A.

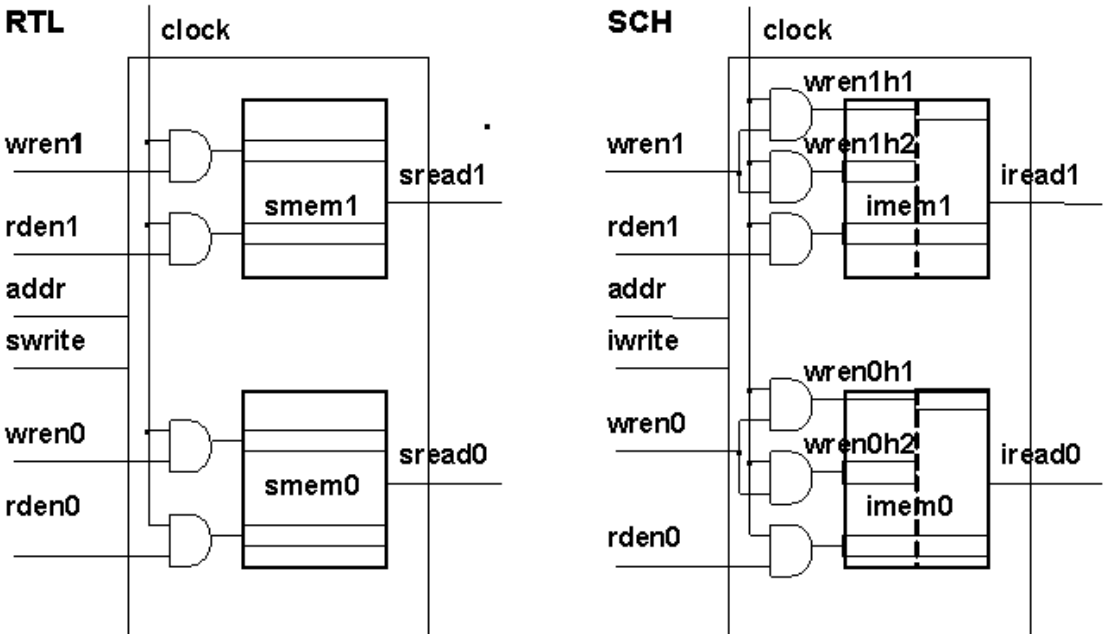
Note that A is not a logical consequence of $A_1, \ldots, A_n$ if and only if the set of formulas $A_1, \ldots, A_n, \neg A$ is satisfiable.

We have to determine whether the fact that Eismann is not a Russian spy follows from the conditions of the puzzle.

Therefore, the problem of solving the puzzle is an instance of the satisfaibility problem.

Circuit Equivalence

Given two circuits, check if they are equivalent. For example:

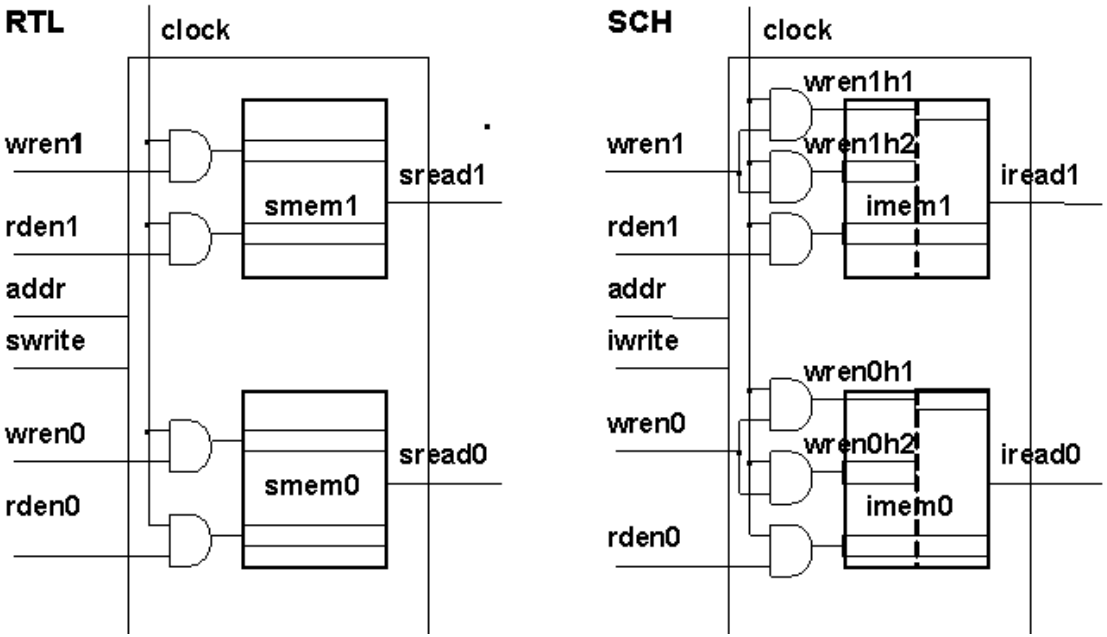


Every circuit is, in fact, a propositional formula.

We know that equivalence-checking for propositional formulas can be reduced to unsatisfiability-checking.

Circuit Equivalence

Given two circuits, check if they are equivalent. For example:

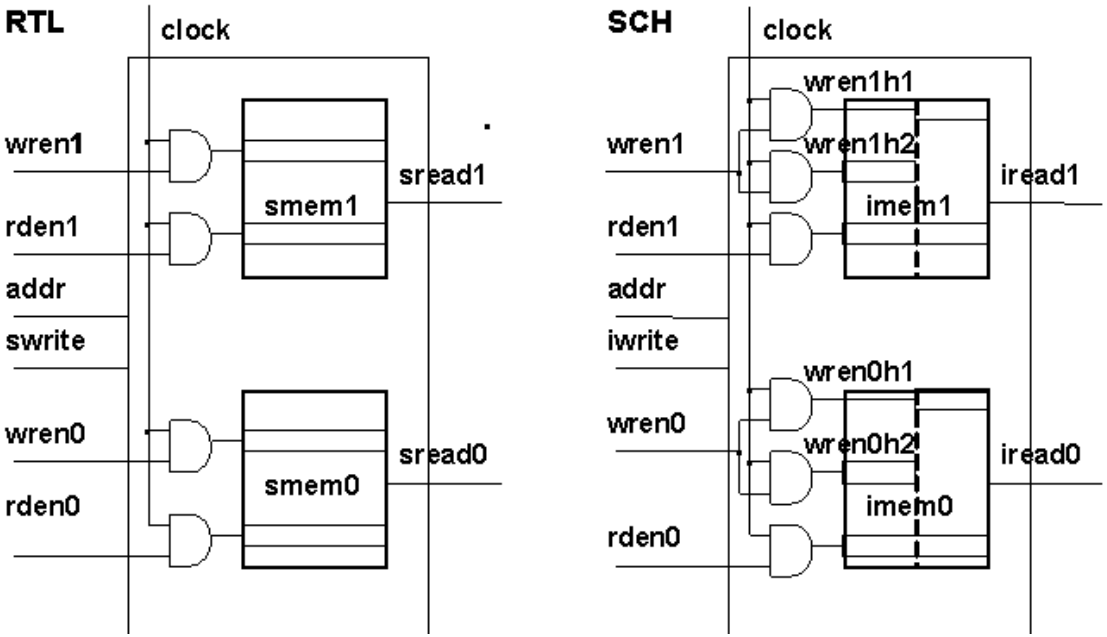


Every circuit is, in fact, a propositional formula.

We know that equivalence-checking for propositional formulas can be reduced to unsatisfiability-checking.

Circuit Equivalence

Given two circuits, check if they are equivalent. For example:



Every circuit is, in fact, a propositional formula.

We know that equivalence-checking for propositional formulas can be reduced to unsatisfiability-checking.

Satisfiability?

Satisfiability checking is a combinatorial problem that is

- easy to formulate;
- hard to solve;
- NP-complete;
- has many algorithms (but only one is commonly used).

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
  - Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
  - Unit clause: $n = 1$.
  - Horn clause: a clause with at most one positive literal.

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
  - Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
  - Unit clause: $n = 1$.
  - Horn clause: a clause with at most one positive literal.

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
  - Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
  - Unit clause: $n = 1$.
  - Horn clause: a clause with at most one positive literal.

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
  - Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
  - Unit clause: $n = 1$.
  - Horn clause: a clause with at most one positive literal.

CNF

- A formula A is in conjunctive normal form, or simply CNF, if it is either $\top$, or $\bot$, or a conjunction of disjunctions of literals:

$$A = \bigwedge_i \bigvee_j L_{i,j}.$$

  (That is, a conjunction of clauses.)
- A formula B is called a conjunctive normal form of a formula A if B is equivalent to A and B is in conjunctive normal form.

Satisfiability on CNF

- An interpretation I satisfies a formula in CNF

$$A = \bigwedge_i \bigvee_j L_{i,j}.$$

  if and only if it satisfies every clause

$$\bigvee_j L_{i,j}.$$

  in it.

- An interpretation I satisfies a clause

$$L_1 \vee \ldots \vee L_k$$

  if and only if it satisfies at least one literal L_m in this clause.

Satisfiability on CNF

- An interpretation I satisfies a formula in CNF

$$A = \bigwedge_i \bigvee_j L_{i,j}.$$

  if and only if it satisfies every clause

$$\bigvee_j L_{i,j}.$$

  in it.
- An interpretation I satisfies a clause

$$L_1 \vee \ldots \vee L_k$$

  if and only if it satisfies at least one literal L_m in this clause.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF transformation

$$
\begin{array}{rcl}
A \leftrightarrow B & \Rightarrow & (\neg A \vee B) \wedge (\neg B \vee A), \\
A \rightarrow B & \Rightarrow & \neg A \vee B, \\
\neg(A \wedge B) & \Rightarrow & \neg A \vee \neg B, \\
\neg(A \vee B) & \Rightarrow & \neg A \wedge \neg B, \\
\neg\neg A & \Rightarrow & A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n & \Rightarrow & (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
 & & \ldots \quad \wedge \\
 & & (A_m \vee B_1 \vee \ldots \vee B_n).
\end{array}
$$

A formula to which no rewrite rule is applicable

- contains no $\leftrightarrow$;
- contains no $\rightarrow$;
- may only contain $\neg$ applied to atoms;
- cannot contain $\wedge$ in the scope of $\vee$;
- (hence) is in CNF.

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\qquad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

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

$$(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n \Rightarrow \begin{array}{l} (A_1 \vee B_1 \vee \ldots \vee B_n) \quad \wedge \\ \qquad \ldots \qquad\qquad\quad \wedge \\ (A_m \vee B_1 \vee \ldots \vee B_n). \end{array}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\qquad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B \quad &\Rightarrow \quad (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B \quad &\Rightarrow \quad \neg A \vee B, \\
\neg(A \wedge B) \quad &\Rightarrow \quad \neg A \vee \neg B, \\
\neg(A \vee B) \quad &\Rightarrow \quad \neg A \wedge \neg B, \\
\neg\neg A \quad &\Rightarrow \quad A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n \quad &\Rightarrow \quad (A_1 \vee B_1 \vee \ldots \vee B_n) \quad \wedge \\
&\qquad\qquad \ldots \qquad\qquad \wedge \\
&\qquad\qquad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

$$\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A), \\
A \to B &\Rightarrow \neg A \vee B, \\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B, \\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B, \\
\neg\neg A &\Rightarrow A, \\
(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n &\Rightarrow (A_1 \vee B_1 \vee \ldots \vee B_n) \wedge \\
&\qquad \ldots \qquad \wedge \\
&\quad (A_m \vee B_1 \vee \ldots \vee B_n).
\end{aligned}$$

CNF, example

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\neg(\neg((p \to q) \wedge (p \wedge q \to r)) \vee (p \to r)) \Rightarrow$$
$$\neg\neg((p \to q) \wedge (p \wedge q \to r)) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(p \to r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg(\neg p \vee r) \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge \neg\neg p \wedge r \Rightarrow$$
$$(p \to q) \wedge (p \wedge q \to r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg(p \wedge q) \vee r) \wedge p \wedge \neg r \Rightarrow$$
$$(p \to q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

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

$$(A_1 \wedge \ldots \wedge A_m) \vee B_1 \vee \ldots \vee B_n \Rightarrow \begin{array}{l} (A_1 \vee B_1 \vee \ldots \vee B_n) \quad \wedge \\ \qquad \ldots \qquad\qquad\quad \wedge \\ (A_m \vee B_1 \vee \ldots \vee B_n). \end{array}$$

CNF and satisfiability

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\ldots$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

Therefore, the formula

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$$

has the same models as the set consisting of four clauses

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r$$
$$p$$
$$\neg r$$

The CNF transformation allows one to reduce the satisfiability problem for formulas to the satisfiability problem for sets of clauses.

CNF and satisfiability

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\ldots$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

Therefore, the formula

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$$

has the same models as the set consisting of four clauses

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r$$
$$p$$
$$\neg r$$

The CNF transformation allows one to reduce the satisfiability problem for formulas to the satisfiability problem for sets of clauses.

CNF and satisfiability

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r)) \Rightarrow$$
$$\ldots$$
$$(\neg p \vee q) \wedge (\neg p \vee \neg q \vee r) \wedge p \wedge \neg r$$

Therefore, the formula

$$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$$

has the same models as the set consisting of four clauses

$$\neg p \vee q$$
$$\neg p \vee \neg q \vee r$$
$$p$$
$$\neg r$$

The CNF transformation allows one to reduce the satisfiability problem for formulas to the satisfiability problem for sets of clauses.

Problem

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

Problem

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \Rightarrow$$
$$(\neg p_1 \vee (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \wedge$$
$$(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \Rightarrow$$
$$(\neg p_1 \vee \quad ((\neg p_2 \vee (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \wedge$$
$$(p_2 \vee \neg(p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))) \wedge$$
$$(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))$$

If we continue, the formula will grow exponentially.

Problem

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

$$\begin{aligned}
&p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \Rightarrow \\
&(\neg p_1 \vee (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \Rightarrow \\
&(\neg p_1 \vee \quad ((\neg p_2 \vee (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))\wedge \\
&\qquad\qquad (p_2 \vee \neg(p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))\wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))
\end{aligned}$$

If we continue, the formula will grow exponentially.

Problem

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

$$\begin{aligned}
&p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \Rightarrow \\
&(\neg p_1 \vee (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \Rightarrow \\
&(\neg p_1 \vee \quad ((\neg p_2 \vee (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))\wedge \\
&\qquad\qquad (p_2 \vee \neg(p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))
\end{aligned}$$

If we continue, the formula will grow exponentially.

Problem

Compute the CNF of

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))).$$

$$\begin{aligned}
&p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))) \Rightarrow \\
&(\neg p_1 \vee (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))) \Rightarrow \\
&(\neg p_1 \vee \quad ((\neg p_2 \vee (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))\wedge \\
&\qquad\qquad (p_2 \vee \neg(p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))) \wedge \\
&(p_1 \vee \neg(p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6)))))
\end{aligned}$$

If we continue, the formula will grow exponentially.

CNF is exponential

There are formulas for which the shortest CNF has exponential size.

Is there any way to avoid exponential blowup?

CNF is exponential

There are formulas for which the shortest CNF has exponential size.

Is there any way to avoid exponential blowup?

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

Idea

Using so-called naming or definition introduction.

- Take a non-trivial subformula A.
- Introduce a new name n for it. A name is a new propositional variable.
- Add a formula stating that n is equivalent to A (definition for n).

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

- Replace the subformula by its name:

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow n)))$$
$$n \leftrightarrow (p_5 \leftrightarrow p_6)$$

The new set of two formulas has the same models as the original one if we restrict ourselves to the original set of variables $\{p_1, \ldots, p_6\}$.
But this set is not equivalent to the original formula.

After several steps

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$

$$\begin{array}{l} p_1 \leftrightarrow (p_2 \leftrightarrow n_3); \\ n_3 \leftrightarrow (p_3 \leftrightarrow n_4); \\ n_4 \leftrightarrow (p_4 \leftrightarrow n_5); \\ n_5 \leftrightarrow (p_5 \leftrightarrow p_6). \end{array}$$

The conversion of the original formula to CNF introduces 32 copies of p_6.

The conversion of the new set of formulas to CNF introduces 4 copies of p_6.

After several steps

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$

$$\begin{array}{l} p_1 \leftrightarrow (p_2 \leftrightarrow n_3); \\ n_3 \leftrightarrow (p_3 \leftrightarrow n_4); \\ n_4 \leftrightarrow (p_4 \leftrightarrow n_5); \\ n_5 \leftrightarrow (p_5 \leftrightarrow p_6). \end{array}$$

The conversion of the original formula to CNF introduces 32 copies of p_6.

The conversion of the new set of formulas to CNF introduces 4 copies of p_6.

After several steps

$$p_1 \leftrightarrow (p_2 \leftrightarrow (p_3 \leftrightarrow (p_4 \leftrightarrow (p_5 \leftrightarrow p_6))))$$

$$\begin{array}{l} p_1 \leftrightarrow (p_2 \leftrightarrow n_3); \\ n_3 \leftrightarrow (p_3 \leftrightarrow n_4); \\ n_4 \leftrightarrow (p_4 \leftrightarrow n_5); \\ n_5 \leftrightarrow (p_5 \leftrightarrow p_6). \end{array}$$

The conversion of the original formula to CNF introduces 32 copies of p_6.

The conversion of the new set of formulas to CNF introduces 4 copies of p_6.

Clausal Form

- Clausal form of a formula A: a set of clauses which is satisfiable if and only if A is satisfiable.
- Clausal form of a set S of formulas: a set of clauses which is satisfiable if and only if so is S.

We can require even more: that A and S have the same models in the language of A.

Using clausal normal forms instead of conjunctive normal forms we can convert any formula to a set of clauses in almost linear time.

Clausal Form

- Clausal form of a formula A: a set of clauses which is satisfiable if and only if A is satisfiable.
- Clausal form of a set S of formulas: a set of clauses which is satisfiable if and only if so is S.

We can require even more: that A and S have the same models in the language of A.

Using clausal normal forms instead of conjunctive normal forms we can convert any formula to a set of clauses in almost linear time.

Clausal Form

- Clausal form of a formula A: a set of clauses which is satisfiable if and only if A is satisfiable.
- Clausal form of a set S of formulas: a set of clauses which is satisfiable if and only if so is S.

We can require even more: that A and S have the same models in the language of A.

Using clausal normal forms instead of conjunctive normal forms we can convert any formula to a set of clauses in almost linear time.

Clausal Form

- Clausal form of a formula A: a set of clauses which is satisfiable if and only if A is satisfiable.
- Clausal form of a set S of formulas: a set of clauses which is satisfiable if and only if so is S.

We can require even more: that A and S have the same models in the language of A.

Using clausal normal forms instead of conjunctive normal forms we can convert any formula to a set of clauses in almost linear time.

Definitional Clause Form Transformation

This algorithm converts a formula A into a set of clauses S such that S is a clausal normal form of A.
If A has the form $C_1 \wedge \ldots \wedge C_n$, where $n \geq 1$ and each C_i is a clause, then $S \overset{\text{def}}{\Leftrightarrow} \{C_1, \ldots, C_n\}$.
Otherwise, introduce a name for each subformula B of A such that B is not a literal and use this name instead of the formula.

Example

	subformula	definition	clauses
			n_1
n_1	$\neg((p \to q) \wedge (p \wedge q \to r) \to (p \to r))$	$n_1 \leftrightarrow \neg n_2$	$\neg n_1 \vee \neg n_2$ $n_1 \vee n_2$
n_2	$(p \to q) \wedge (p \wedge q \to r) \to (p \to r)$	$n_2 \leftrightarrow (n_3 \to n_7)$	$\neg n_2 \vee \neg n_3 \vee n_7$ $n_3 \vee n_2$ $\neg n_7 \vee n_2$
n_3	$(p \to q) \wedge (p \wedge q \to r)$	$n_3 \leftrightarrow (n_4 \wedge n_5)$	$\neg n_3 \vee n_4$ $\neg n_3 \vee n_5$ $\neg n_4 \vee \neg n_5 \vee n_3$
n_4	$p \to q$	$n_4 \leftrightarrow (p \to q)$	$\neg n_4 \vee \neg p \vee q$ $p \vee n_4$ $\neg q \vee n_4$
n_5	$p \wedge q \to r$	$n_5 \leftrightarrow (n_6 \to r)$	$\neg n_5 \vee \neg n_6 \vee r$ $n_6 \vee n_5$ $\neg r \vee n_5$
n_6	$p \wedge q$	$n_6 \leftrightarrow (p \wedge q)$	$\neg n_6 \vee p$ $\neg n_6 \vee q$ $\neg p \vee \neg q \vee n_6$
n_7	$p \to r$	$n_7 \leftrightarrow (p \to r)$	$\neg n_7 \vee \neg p \vee r$ $p \vee n_7$ $\neg r \vee n_7$

Outline

Binary Resolution Inference System

The binary resolution inference system, denoted by $\mathbb{BR}$, consists of two inference rules:

- Binary resolution, denoted by BR

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Binary Resolution Inference System

The binary resolution inference system, denoted by $\mathbb{BR}$, consists of two inference rules:

- Binary resolution, denoted by BR

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Binary Resolution Inference System

The binary resolution inference system, denoted by $\mathbb{BR}$, consists of two inference rules:

- Binary resolution, denoted by BR

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Inference System

- inference has the form

$$\frac{F_1 \quad \ldots \quad F_n}{G} \ ,$$

  where $n \geq 0$ and $F_1, \ldots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \ldots, F_n$ are called its premises.
- An inference rule R is a set of inferences.
- Every inference $I \in R$ is called an instance of R.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Inference System

- inference has the form

$$\frac{F_1 \quad \ldots \quad F_n}{G} \ ,$$

  where $n \geq 0$ and $F_1, \ldots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \ldots, F_n$ are called its premises.
- An inference rule R is a set of inferences.
- Every inference $I \in R$ is called an instance of R.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Inference System

- inference has the form

$$\frac{F_1 \quad \ldots \quad F_n}{G} \ ,$$

  where $n \geq 0$ and $F_1, \ldots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \ldots, F_n$ are called its premises.
- An inference rule R is a set of inferences.
- Every inference $I \in R$ is called an instance of R.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Inference System

- inference has the form

$$\frac{F_1 \quad \ldots \quad F_n}{G} \ ,$$

  where $n \geq 0$ and $F_1, \ldots, F_n, G$ are formulas.
- The formula G is called the conclusion of the inference;
- The formulas $F_1, \ldots, F_n$ are called its premises.
- An inference rule R is a set of inferences.
- Every inference $I \in R$ is called an instance of R.
- An Inference system $\mathbb{I}$ is a set of inference rules.
- Axiom: inference rule with no premises.

Derivation, Proof

- Derivation in an inference system $\mathbb{I}$: a tree built from inferences in $\mathbb{I}$.
- If the root of this derivation is E, then we say it is a derivation of E.
- Proof of E: a finite derivation whose leaves are axioms.
- Derivation of E from $E_1, \ldots, E_m$: a finite derivation of E whose every leaf is either an axiom or one of the expressions $E_1, \ldots, E_m$.

Soundness

- An inference is sound if the conclusion of this inference is a logical consequence of its premises.
- An inference rule is sound if every inference of this rule is sound.
- An inference system is sound if every inference rule in this system is sound.

Theorem

$\mathbb{BR}$ *is sound.*

Soundness

- An inference is sound if the conclusion of this inference is a logical consequence of its premises.
- An inference rule is sound if every inference of this rule is sound.
- An inference system is sound if every inference rule in this system is sound.

Theorem

$\mathbb{BR}$ *is sound.*

Consequence of Soundness

Theorem

Let S be a set of clauses. If $\square$ can be derived from S in $\mathbb{BR}$, then S is unsatisfiable.

Example

Consider the following set of clauses

$$\{\neg p \vee \neg q,\ \neg p \vee q,\ p \vee \neg q,\ p \vee q\}.$$

The following derivation derives the empty clause from this set:

$$\dfrac{\dfrac{\dfrac{p \vee q \quad p \vee \neg q}{p \vee p}\ (\text{BR})}{p}\ (\text{Fact}) \qquad \dfrac{\dfrac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p \vee \neg p}\ (\text{BR})}{\neg p}\ (\text{Fact})}{\square}\ (\text{BR})$$

Hence, this set of clauses is unsatisfiable.

Example

Consider the following set of clauses

$$\{\neg p \vee \neg q,\ \neg p \vee q,\ p \vee \neg q,\ p \vee q\}.$$

The following derivation derives the empty clause from this set:

$$\dfrac{\dfrac{\dfrac{p \vee q \quad p \vee \neg q}{p \vee p}\,(\text{BR})}{p}\,(\text{Fact}) \qquad \dfrac{\dfrac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p \vee \neg p}\,(\text{BR})}{\neg p}\,(\text{Fact})}{\square}\,(\text{BR})$$

Hence, this set of clauses is unsatisfiable.

Example

Consider the following set of clauses

$$\{\neg p \vee \neg q,\ \neg p \vee q,\ p \vee \neg q,\ p \vee q\}.$$

The following derivation derives the empty clause from this set:

$$\dfrac{\dfrac{\dfrac{p \vee q \quad p \vee \neg q}{p \vee p}\,(\text{BR})}{p}\,(\text{Fact}) \qquad \dfrac{\dfrac{\neg p \vee q \quad \neg p \vee \neg q}{\neg p \vee \neg p}\,(\text{BR})}{\neg p}\,(\text{Fact})}{\square}\,(\text{BR})$$

Hence, this set of clauses is unsatisfiable.

Writing derivations in the linear form

(1)	$\neg p \vee \neg q$	input	
(2)	$\neg p \vee q$	input	
(3)	$p \vee \neg q$	input	
(4)	$p \vee q$	input	
(5)	$\neg p \vee \neg p$	BR	(1,2)
(6)	$\neg p$	Fact	(5)
(7)	$p \vee p$	BR	(3,4)
(8)	p	Fact	(7)
(9)	$\square$	BR	(6,8)

Completeness

$\mathbb{BR}$ is complete, that is, if a set of clauses is unsatisfiable, then one can derive an empty clause from this set.

Selection Function

The binary resolution inference system has too many inferences. There are restrictions on resolution that allow for fewer inferences but preserve completeness.

To define these systems we need a new notion.

A literal selection function selects one or more literals in every non-empty clause.

We will sometimes denote selected literals by underlining them, e.g.,

$$\underline{p} \vee \neg q$$

Selection Function

The binary resolution inference system has too many inferences. There are restrictions on resolution that allow for fewer inferences but preserve completeness.

To define these systems we need a new notion.

A literal selection function selects one or more literals in every non-empty clause.

We will sometimes denote selected literals by underlining them, e.g.,

$$\underline{p} \vee \neg q$$

Selection Function

The binary resolution inference system has too many inferences. There are restrictions on resolution that allow for fewer inferences but preserve completeness.

To define these systems we need a new notion.

A literal selection function selects one or more literals in every non-empty clause.

We will sometimes denote selected literals by underlining them, e.g.,

$$\underline{p} \vee \neg q$$

Selection Function

The binary resolution inference system has too many inferences. There are restrictions on resolution that allow for fewer inferences but preserve completeness.

To define these systems we need a new notion.

A literal selection function selects one or more literals in every non-empty clause.

We will sometimes denote selected literals by underlining them, e.g.,

$$\underline{p} \vee \neg q$$

Binary Resolution with Selection

The binary resolution inference system, denoted by $\mathbb{BR}_\sigma$, consists of two inference rules:

- Binary resolution denoted by BR

$$\frac{\underline{p} \vee C_1 \quad \underline{\neg p} \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Binary resolution with selection is incomplete.

However, it is complete for some well-behaved selection functions.

Binary Resolution with Selection

The binary resolution inference system, denoted by $\mathbb{BR}_\sigma$, consists of two inference rules:

- Binary resolution denoted by BR

$$\frac{\underline{p} \vee C_1 \quad \underline{\neg p} \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Binary resolution with selection is incomplete.

However, it is complete for some well-behaved selection functions.

Binary Resolution with Selection

The binary resolution inference system, denoted by $\mathbb{BR}_\sigma$, consists of two inference rules:

- Binary resolution denoted by BR

$$\frac{\underline{p} \vee C_1 \quad \underline{\neg p} \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

- Factoring, denoted by Fact:

$$\frac{L \vee L \vee C}{L \vee C} \text{ (Fact).}$$

Binary resolution with selection is incomplete.

However, it is complete for some well-behaved selection functions.

Unrestricted binary resolution and binary resolution with selection

Consider the selection function that selects all literals in a clause. Then the binary resolution rule:

$$\frac{p \vee C_1 \quad \neg p \vee C_2}{C_1 \vee C_2} \text{ (BR).}$$

becomes a special case of binary resolution with selection.

Literal Orderings

Consider any total ordering $\succ$ on propositional variables. We want to extend it to literals.

Let $L_1 = (\neg)A_1$ and $L_2 = (\neg)A_2$ be literals. We let $L_1 \succ_{lit} L_2$ if and only if one of the following conditions holds:

1. $A_1 \succ A_2$; or
2. $A_1 = A_2$, L_1 is negative and L_2 is positive.

In other words, we compare literals by first comparing the atoms of these literals and if the atoms are equal define the negative literal to be greater.

Literal Orderings

Consider any total ordering $\succ$ on propositional variables. We want to extend it to literals.

Let $L_1 = (\neg)A_1$ and $L_2 = (\neg)A_2$ be literals. We let $L_1 \succ_{lit} L_2$ if and only if one of the following conditions holds:

1. $A_1 \succ A_2$; or
2. $A_1 = A_2$, L_1 is negative and L_2 is positive.

In other words, we compare literals by first comparing the atoms of these literals and if the atoms are equal define the negative literal to be greater.

Literal Orderings

Consider any total ordering $\succ$ on propositional variables. We want to extend it to literals.

Let $L_1 = (\neg)A_1$ and $L_2 = (\neg)A_2$ be literals. We let $L_1 \succ_{lit} L_2$ if and only if one of the following conditions holds:

1. $A_1 \succ A_2$; or
2. $A_1 = A_2$, L_1 is negative and L_2 is positive.

In other words, we compare literals by first comparing the atoms of these literals and if the atoms are equal define the negative literal to be greater.

Ordered resolution

Fix an ordering $\succ$ on the set of propositional variables and let $\succ_{lit}$ be corresponding literal ordering. Consider the selection function σ that selects all maximal w.r.t. $\succ_{lit}$ literals.

Theorem

$\mathbb{BR}_\sigma$ is complete, that is, for every unsatisfiable set of clauses S one can derive the empty clause from clauses in S using inferences in $\mathbb{BR}_\sigma$.

Ordered resolution

Fix an ordering $\succ$ on the set of propositional variables and let $\succ_{lit}$ be corresponding literal ordering. Consider the selection function σ that selects all maximal w.r.t. $\succ_{lit}$ literals.

Theorem

$\mathbb{BR}_\sigma$ *is complete, that is, for every unsatisfiable set of clauses* S *one can derive the empty clause from clauses in* S *using inferences in* $\mathbb{BR}_\sigma$.

Ordered resolution: example

Assume $q \succ p$.

(1)	$\neg p \vee \underline{\neg q}$	input	
(2)	$\neg p \vee \underline{q}$	input	
(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	
(5)	$\underline{\neg p} \vee \underline{\neg p}$	BR	(1,2)
(6)	$\underline{p} \vee \underline{p}$	BR	(3,4)
(7)	$\underline{p}$	Fact	(6)
(8)	$\underline{\neg p}$	BR	(6,7)
(9)	$\square$	BR	(6,8)

Note: fewer inferences are enabled compared to unrestricted binary resolution.

Ordered resolution: example

Assume $q \succ p$.

(1)	$\neg p \vee \underline{\neg q}$	input	
(2)	$\neg p \vee \underline{q}$	input	
(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	
(5)	$\underline{\neg p} \vee \underline{\neg p}$	BR	(1,2)
(6)	$\underline{p} \vee \underline{p}$	BR	(3,4)
(7)	$\underline{p}$	Fact	(6)
(8)	$\underline{\neg p}$	BR	(6,7)
(9)	$\square$	BR	(6,8)

Note: fewer inferences are enabled compared to unrestricted binary resolution.

End of Lecture 1

Slides for lecture 1 ended here . . .

Inference Process

Inference process: sequence of sets of clauses $S_0, S_1, \ldots$, denoted by

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$$

$(S_i \Rightarrow S_{i+1})$ is a step of this process.

We say that this step is an $\mathbb{I}$-step if

1. there exists an inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

in $\mathbb{I}$ such that $\{C_1, \ldots, C_n\} \subseteq S_i$;

2. $S_{i+1} = S_i \cup \{C\}$.

An $\mathbb{I}$-inference process is an inference process whose every step is an $\mathbb{I}$-step.

Inference Process

Inference process: sequence of sets of clauses $S_0, S_1, \ldots$, denoted by

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$$

$(S_i \Rightarrow S_{i+1})$ is a step of this process.

We say that this step is an $\mathbb{I}$-step if

1. there exists an inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

   in $\mathbb{I}$ such that $\{C_1, \ldots, C_n\} \subseteq S_i$;
2. $S_{i+1} = S_i \cup \{C\}$.

An $\mathbb{I}$-inference process is an inference process whose every step is an $\mathbb{I}$-step.

Inference Process

Inference process: sequence of sets of clauses $S_0, S_1, \ldots$, denoted by

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$$

$(S_i \Rightarrow S_{i+1})$ is a step of this process.

We say that this step is an $\mathbb{I}$-step if

1. there exists an inference

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

   in $\mathbb{I}$ such that $\{C_1, \ldots, C_n\} \subseteq S_i$;
2. $S_{i+1} = S_i \cup \{C\}$.

An $\mathbb{I}$-inference process is an inference process whose every step is an $\mathbb{I}$-step.

Property

Lemma

Let $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be an $\mathbb{I}$-inference process and a clause C belongs to some S_i. Then S_i is derivable in $\mathbb{I}$ from S_0.

Can we prove the inverse?

Property

Lemma

Let $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be an $\mathbb{I}$-inference process and a clause C belongs to some S_i. Then S_i is derivable in $\mathbb{I}$ from S_0.

Can we prove the inverse?

Limit and Fairness

The limit of an inference process $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ is the set of clauses $\bigcup_i S_i$.

Let $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be an inference process with the limit S_∞. The process is called fair if for every $\mathbb{I}$-inference

$$\frac{C_1 \quad \ldots \quad C_n}{C} \ ,$$

if $\{C_1, \ldots, C_n\} \subseteq S_\infty$, then there exists i such that $C \in S_i$.

Limit and Fairness

The limit of an inference process $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ is the set of clauses $\bigcup_i S_i$.

Let $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be an inference process with the limit S_∞. The process is called fair if for every $\mathbb{I}$-inference

$$\frac{C_1 \quad \ldots \quad C_n}{C} \ ,$$

if $\{C_1, \ldots, C_n\} \subseteq S_\infty$, then there exists i such that $C \in S_i$.

Completeness, reformulated

Theorem Let $\mathbb{I}$ be an inference system. The following conditions are equivalent.

1. $\mathbb{I}$ is complete.
2. For every unsatisfiable set of clauses S_0 and any fair $\mathbb{I}$-inference process with the initial set S_0, the limit of this inference process contains $\square$.

Saturated Set of Clauses

Let $\mathbb{I}$ be an inference system and S be a set of clauses. S is called saturated with respect to $\mathbb{I}$, or simply $\mathbb{I}$-*saturated*, if for every inference of $\mathbb{I}$ with premises in S, the conclusion of this inference also belongs to S.

The closure of S with respect to $\mathbb{I}$, or simply $\mathbb{I}$-closure, is the smallest set S' containing S and saturated with respect to $\mathbb{I}$.

Completeness of Ordered Resolution

Theorem (Completeness)

Take any well-founded ordering $\succ$ and consider the selection function σ that selects all maximal w.r.t. $\succ_{lit}$ literals. Let S_0 be a set of clauses and $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be a fair $\mathbb{BR}_\sigma$-inference process. Then S_0 is unsatisfiable if and only if $\square \in S_i$ for some i.

Lemma

The limit S_ω is saturated.

Lemma

The limit S_ω is logically equivalent to the initial set S_0.

Lemma

A saturated set S of clauses is unsatisfiable if and only if $\square \in S$.

Completeness of Ordered Resolution

Theorem (Completeness)

Take any well-founded ordering $\succ$ and consider the selection function σ that selects all maximal w.r.t. $\succ_{lit}$ literals. Let S_0 be a set of clauses and $S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$ be a fair $\mathbb{BR}_\sigma$-inference process. Then S_0 is unsatisfiable if and only if $\square \in S_i$ for some i.

Lemma

The limit S_ω is saturated.

Lemma

The limit S_ω is logically equivalent to the initial set S_0.

Lemma

A saturated set S of clauses is unsatisfiable if and only if $\square \in S$.

Corollaries

Completeness of Binary Resolution. Binary resolution is complete.
Compactness. Let S be a countably infinite set of clauses. Then S is unsatisfiable if and only if it contains a finite unsatisfiable subset.
Note. The assumption of being countably infinite can be dropped.

Problem: search space grows too fast

Idea: remove some clauses from the search space.
We will consider later how clauses can be removed without compromising completeness.

Inference Process with Deletion

Let $\mathbb{I}$ be an inference system. Consider an inference process with two kinds of step $S_i \Rightarrow S_{i+1}$:

1. $\mathbb{I}$-inference;
2. deletion of a clause in S_i, that is

$$S_{i+1} = S_i - \{C\},$$

where $C \in S_i$.

Fairness: Persistent Clauses and Limit

Consider an inference process

$$S_0 \Rightarrow S_1 \Rightarrow S_2 \Rightarrow \ldots$$

A clause C is called persistent if

$$\exists i \forall j \geq i (C \in S_j).$$

The limit S_ω of the inference process is the set of all persistent clauses:

$$S_\omega = \bigcup_{i=0,1,\ldots} \bigcap_{j \geq i} S_j.$$

Fairness

The process is called $\mathbb{I}$-fair if every inference with persistent premises in S_ω has been applied, that is, if

$$\frac{C_1 \quad \ldots \quad C_n}{C}$$

is an inference in $\mathbb{I}$ and $\{C_1, \ldots, C_n\} \subseteq S_\omega$, then $C \in S_i$ for some i.

Deletion rules

Tautology: a clause of the form $p \vee \neg p \vee C$. Tautology deletion: deletion of tautologies from the search space.
Finite multiset: like a set but elements may occur more than once. Example: $\{1, 2, 2, 5, 5, 5\}$. A clause can be considered as a multiset of its literals.
A clause C_1 is said to subsume any clause $C_1 \vee C_2$, where C_2 is non-empty. In other words, C_1 subsumes C_2 if and only if C_1 is a submultiset of C_2.
Subsumption deletion: deletion of subsumed clauses from the search space.

Completeness with deletion rules

Subsumption and tautology deletion does not compromise completeness of binary and ordered resolution.
That is, for every fair inference process with subsumption a tautology deletion, if the initial set of clauses is unsatisfiable, then the limit of the process contains the empty clause.

Example: inference process with deletion

(1)	$\neg p \vee \underline{\neg q}$	input	
(2)	$\neg p \vee \underline{q}$	input	
(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	

(1)	$\neg p \vee \underline{\neg q}$	input	
(2)	$\neg p \vee \underline{q}$	input	
(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	
(5)	$\underline{\neg p} \vee \underline{\neg p}$	BR	(1,2)

(1)	$\neg p \vee \underline{\neg q}$	input	
(2)	$\neg p \vee \underline{q}$	input	
(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	
(5)	$\underline{\neg p} \vee \underline{\neg p}$	BR	(1,2)
(6)	$\underline{\neg p}$	Fact	(5)

(3)	$p \vee \underline{\neg q}$	input	
(4)	$p \vee \underline{q}$	input	
(6)	$\underline{\neg p}$	Fact	(5)

Example: inference process with deletion

(3)	$p \vee \neg q$	input	
(4)	$p \vee q$	input	
(6)	$\neg p$	Fact	(5)

(3)	$p \vee \neg q$	input	
(4)	$p \vee q$	input	
(6)	$\neg p$	Fact	(5)
(7)	$p \vee p$	BR	(3,4)

(3)	$p \vee \neg q$	input	
(4)	$p \vee q$	input	
(6)	$\neg p$	Fact	(5)
(7)	$p \vee p$	BR	(3,4)
(8)	p	Fact	(7)

(6)	$\neg p$	Fact	(5)
(8)	p	Fact	(7)

(6)	$\neg p$	Fact	(5)
(8)	p	Fact	(7)
(9)	$\square$	BR	(6,8)

Outline

Prolog

Running Prolog at the school:

1. boot a computer using Linux;
2. start a terminal;
3. type `sicstus` in it

and the Sictus Prolog will start.
Running Prolog on your home computer/laptop:

- Download SWI Prolog and follow instructions.

Example of a Prolog program

- Terms:
  - constants, variables (start with an upper-case letter or an underscore);
  - compound: name(arg_1,. . . ,arg_k);
  - Ground terms: terms with no variables
- Clauses:
  - Rules: Head :- $Goal_1$, . . . , $Goal_k$.
  - Facts: Head.
    i.e. a rule without any goals or body
  - Goals: :- $Goal_1$, . . . , $Goal_k$.
    i.e. a rule without a head.

Examples of clauses

```
parent(john,juliet).
:- parent(john,X).
parent(X,juliet).
greater_than(succ(X),zero).
```

Predicate definition: collection of rules with the same predicate (head name).

```
ancestor(X,Y) :- mother(X,Y).
ancestor(X,Y) :- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
...
```

Examples of clauses

```
parent(john,juliet).
:- parent(john,X).
parent(X,juliet).
greater_than(succ(X),zero).
```

Predicate definition: collection of rules with the same predicate (head name).

```
ancestor(X,Y) :- mother(X,Y).
ancestor(X,Y) :- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
...
```

Examples of clauses

```
parent(john,juliet).
:- parent(john,X).
parent(X,juliet).
greater_than(succ(X),zero).
```

Predicate definition: collection of rules with the same predicate (head name).

```
ancestor(X,Y) :- mother(X,Y).
ancestor(X,Y) :- father(X,Y).
ancestor(X,Y) :- ancestor(X,Z), ancestor(Z,Y).
...
```

Rules

Meaning of a rule `Head :- Goal`$_1$`,...,Goal`$_n$:

- If `Goal`$_1$ and `Goal`$_2$ and ... and `Goal`$_k$ all hold, then `Head` holds.

Program: a sequence of clauses.

```
ancestor(X,Y) :- father(X,Y).
father(X,Y) :- parent(X,Y), male(X).
parent(john,juliet).
male(john).
```

Queries:

```
:- ancestor(john,juliet).
:- father(john,juliet).
:- parent(john,juliet),male(john).
```

Rules

Meaning of a rule `Head :- Goal`$_1$`,...,Goal`$_n$:

- If `Goal`$_1$ and `Goal`$_2$ and ... and `Goal`$_k$ all hold, then `Head` holds.

Program: a sequence of clauses.

```
ancestor(X,Y) :- father(X,Y).
father(X,Y) :- parent(X,Y), male(X).
parent(john,juliet).
male(john).
```

Queries:

```
:- ancestor(john,juliet).
:- father(john,juliet).
:- parent(john,juliet),male(john).
```

Rules

Meaning of a rule `Head :- Goal`$_1$`,...,Goal`$_n$:

- If `Goal`$_1$ and `Goal`$_2$ and ... and `Goal`$_k$ all hold, then `Head` holds.

Program: a sequence of clauses.

```
ancestor(X,Y) :- father(X,Y).
father(X,Y) :- parent(X,Y), male(X).
parent(john,juliet).
male(john).
```

Queries:

```
:- ancestor(john,juliet).
:- father(john,juliet).
:- parent(john,juliet),male(john).
```

Prolog program with recursion

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
parent(chaz,john).
parent(john,juliet).
:- ancestor(chaz,juliet).
:- parent(chaz,john),ancestor(john,juliet).
:- ancestor(john,juliet).
:- parent(john,juliet).
```

Goal with variables: find a substitution for the variables that makes this goal derivable:

```
:- ancestor(chaz,X).
```

Prolog program with recursion

```
ancestor(X,Y) :- parent(X,Y).
ancestor(X,Y) :- parent(X,Z),ancestor(Z,Y).
parent(chaz,john).
parent(john,juliet).
:- ancestor(chaz,juliet).
:- parent(chaz,john),ancestor(john,juliet).
:- ancestor(john,juliet).
:- parent(john,juliet).
```

Goal with variables: find a substitution for the variables that makes this goal derivable:

```
:- ancestor(chaz,X).
```

How does Prolog answer goals?

Search Strategy: process subgoals left-to-right, top-to-bottom (but see later. . .)

Use the trace facility of Prolog.

How does Prolog answer goals?

Search Strategy: process subgoals left-to-right, top-to-bottom (but see later. . .)

Use the trace facility of Prolog.

Built-in predicates

Built-in predicates which perform evaluation:

- operators: `+`, `*`, `-`, `/`
- comparison: `<`, `>`, `<=`, `>=`
- equality, inequality: `=`, `==`, `\==`
  - `X = 2 * 3 * 7`
  - `42 = 2 * 3 * 7`
- invoke evaluation:
  - `42 is 2 * 3 * 7`
  - `X is 2 * 3 * 7`

Outline

Propositional Logic and Infinite Domains

Consider simple propositions:

1. The successor of 0 is greater than 0;
2. The successor of 1 is greater than 1;
3. The successor of 2 is greater than 2;
4. The successor of 3 is greater than 3;
5. The successor of 4 is greater than 4;
6. ...

We can use them in propositional logic. But how can we express the property

- The successor of every natural number is greater than this number?

To express it we need a conjunction of an infinite number of propositions.

First order logic

In first order logic we can express properties of the form for all . . . and there exists . . . using quantifiers.
For example, to express the successor property we can

- introduce a function symbol $succ$ to represent the successor function, so that $succ(x)$ denotes the successor of x;
- introduce a predicate symbol $>$ to represent the order on numbers, so that $x > y$ denotes that x is greater than y;
- use a quantifier $\forall$ to express that the successor of every number is greater than this number as $(\forall x)(succ(x) > x)$.

Syntax: the Language

Signature Σ: a set of

- constants;
- function symbols;
- predicate symbols.

Each function symbol and predicate symbol has an associated arity (the number of arguments).
In addition to elements of the signature, the language will use a countably infinite set of variables.
Example: $succ(x) > x$, here

- x is a variable;
- $succ$ is a function symbol of arity 1 (unary function symbol);
- $>$ is a predicate symbol of arity 2 (binary predicate symbol).

Predicate symbols are sometime called relation symbols.

Syntax: Terms

For convenience we fix a signature.

Term:

- every variable is a term;
- every constant is a term;
- if f is a function symbol of arity n and $t_1, \ldots, t_n$ are terms, then $f(t_1, \ldots, t_n)$ is a term.

Examples:

- x;
- 0;
- $succ(succ(x))$;
- $x + y$ (here the binary function symbol $+$ is written in the infix notation).

Two notations

Prolog notation:

- Variables start with upper-case letters: `X`, `Man`.
- Constants start with lower-case letters: `x`, `man`.

Math notation:

- Variables: x, y, z, u, v, w.
- Constants: a, b, c, d, e.

First-Order Formula

- If p is a predicate symbol of arity n and $t_1, \ldots, t_n$ are terms, then $p(t_1, \ldots, t_n)$ is a formula, also called an atomic formula, or simply atom.
- $\top$nd $\bot$ are formulas.
- If $A_1, \ldots, A_n$ are formulas, where $n \geq 2$, then $(A_1 \wedge \ldots \wedge A_n)$ and $(A_1 \vee \ldots \vee A_n)$ are formulas.
- If A is a formula, then $(\neg A)$ is a formula.
- If A and B are formulas, then $(A \rightarrow B)$ and $(A \leftrightarrow B)$ are formulas.
- If A is a formula and x is a variable then $(\forall x)A$ and $(\exists x)A$ are formulas.

Quantifiers

$(\forall x)A$: A holds for all x.
$(\exists x)A$: A holds for some x or there exists some x such that A.
$\forall$: universal quantifier.
$\exists$: existential quantifier.

Free and Bound Variables

Variable Binding:

- x is bound in $\forall x\ F$ or $\exists x\ F$.
- F is the scope of x
- A variable which is not bound is free.

A formula with no free variables is called closed.

Semantics: Structure

Structure $M = (D, R, F, C)$:

- domain D (non-empty)
- R: assign k-ary relation P^M on D to each k-ary predicate symbol P of L;
- F: assign k-ary function f^M on D to each k-ary function symbol f of L;
- C: assign element a^M from D to each constant symbol a of L.

Value assignment s over M: maps variables to domain elements, that is, $s(x) \in D$.

Values for Terms

t is given value $t^{M,s} \in D$:

Term	Value in D
constant a	$a^{M,s} = a^M$
variable x	$x^{M,s} = s(x)$
n-ary function f	$f(t_1, t_2, \ldots, t_n)^{M,s} = f^M(t_1^{M,s}, t_2^{M,s}, \ldots, t_n^{M,s})$ ($t_1, \ldots, t_n$ are terms)

Notation

Define

$$s[x \leftarrow d](y) \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} s(x), & \text{if } x \neq y; \\ d, & \text{if } x = y. \end{cases}$$

Semantics: Truth

Truth values for formulae: $v_{M,s}(A) \in \{1, 0\}$

Logical Symbol	Truth Value
constant $\top$	$v_{M,s}(\top) = 1$
constant $\bot$	$v_{M,s}(\bot) = 0$
predicate	$v_{M,s}(P(t_1, \ldots, t_n)) = 1$ iff $(t_1^{M,s}, \ldots, t_n^{M,s}) \in P^M$
connective (e.g)	$v_{M,s}(F \wedge G) = 1$ iff $v_{M,s}(F) = 1$ and $v_{M,s}(G) = 1$
quantifier $\forall$	$v_{M,s}(\forall x F) = 1$ iff for all $d \in D$, $v_{M,s[x \leftarrow d]}(F) = 1$
quantifier $\exists$	$v_{M,s}(\exists x F) = 1$ iff for some $d \in D$, $v_{M,s[x \leftarrow d]}(F) = 1$

If $v_{M,s}(F) = 1$, write $M, s \models F$

Satisfiability and validity

If A is closed, then $v_{M,s}(A)$ is independent of s; so we write $M \models A$ and say that is true in M.

- If a formula A is true in M we say that M satisfies A and that M is a model of A, denoted by $M \models A$.
- A is satisfiable (valid) if it is true in some (every) structure.
- Two formulas A and B are called equivalent, denoted $A \equiv B$ if they have the same models.

Example

$A = \forall x \forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))$

Take the tructure $M = (people, \{q \mapsto ancestor, p \mapsto parent\}, \emptyset, \emptyset)$ and any value assignment s:

- $v_{M,s}(\forall x \forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))))) = 1$ iff
- for all $d \in D$,
  $v_{M,s[x \leftarrow d]}(\forall y (q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))))) = 1$ iff
- for all $d \in D$, for all $d' \in D$,
  $v_{M,s[x \leftarrow d][y \leftarrow d']}(q(x,y) \rightarrow (p(x,y) \vee \exists z (p(x,z) \wedge q(z,y)))) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $v_{M,s[x \mapsto d][y \mapsto d']}(q(x,y)) = 1$
  then $v_{M,s[x \mapsto d][y \mapsto d']}(p(x,y) \vee \exists z (p(x,z) \wedge q(z,y))) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $(d,d') \in ancestor$ then either
  $v_{M,s[x \mapsto d][y \mapsto d']}(p(x,y))$ or $v_{M,s[x \mapsto d][y \mapsto d']}(\exists z (p(x,z) \wedge q(z,y)))$ iff
- for all $d \in D$, for all $d' \in D$, if $(d,d') \in ancestor$ then either
  $(d,d') \in parent$ or there exists a $d'' \in D$, such that
  $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(p(x,z) \wedge q(z,y)) = 1$ iff
- for all $d \in D$, for all $d' \in D$, if $(d,d') \in ancestor$ then either
  $(d,d') \in parent$ or there exists a $d'' \in D$, such that
  $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(p(x,z)) = 1$ and $v_{M,s[x \mapsto d][y \mapsto d'][z \mapsto d'']}(q(z,y)) = 1$
  iff
- for all $d \in D$, for all $d' \in D$, if $(d,d') \in ancestor$ then either
  $(d,d') \in parent$ or there exists a $d'' \in D$, such that $(d,d'') \in parent$ and

Literal, clause

- Literal: either an atom p (positive literal) or its negation $\neg p$ (negative literal).
- The complementary literal to L:

$$\overline{L} \stackrel{\text{def}}{\Leftrightarrow} \begin{cases} \neg L, & \text{if } L \text{ is positive;} \\ p, & \text{if } L \text{ has the form } \neg p. \end{cases}$$

  In other words, p and $\neg p$ are complementary.
- Clause: a disjunction $L_1 \vee \ldots \vee L_n$, $n \geq 0$ of literals.
- Empty clause, denoted by $\square$: $n = 0$ (the empty clause is false in every interpretation).
- Unit clause: $n = 1$.

When we consider clauses we assume that the order of literals in them is irrelevant.

Negation Normal Form

A formula A is in negation normal form, or simply NNF, if it is either $\top$, or $\bot$, or is built from literals using only $\wedge$, $\vee$, $\forall$ and $\exists$.

A formula B is called a negation normal form of a formula A if B is equivalent to A and B is in negation normal form.

Negation Normal Form

A formula A is in negation normal form, or simply NNF, if it is either $\top$, or $\bot$, or is built from literals using only $\wedge$, $\vee$, $\forall$ and $\exists$.

A formula B is called a negation normal form of a formula A if B is equivalent to A and B is in negation normal form.

NNF transformation

$$
\begin{aligned}
A \leftrightarrow B &\Rightarrow (\neg A \vee B) \wedge (\neg B \vee A),\\
A \rightarrow B &\Rightarrow \neg A \vee B,\\
\neg(A \wedge B) &\Rightarrow \neg A \vee \neg B,\\
\neg(A \vee B) &\Rightarrow \neg A \wedge \neg B,\\
\neg(\forall x)A &\Rightarrow (\exists x)\neg A,\\
\neg(\exists x)A &\Rightarrow (\forall x)\neg A,\\
\neg\neg A &\Rightarrow A
\end{aligned}
$$

Rectified formulas

Rectified formula F:

- no variable appears both free and bound in F;
- for every variable x, the formula F contains at most one occurrence of quantifiers $\forall x$ or $\exists x$.

Any formula can be transformed into a rectified formula by renaming bound variables.

Rectification: Example

$$p(x) \rightarrow \exists x(p(x) \wedge \forall x(p(x) \vee r \rightarrow \neg p(x))) \Rightarrow$$
$$p(x) \rightarrow \exists x_1(p(x_1) \wedge \forall x(p(x) \vee r \rightarrow \neg p(x))) \Rightarrow$$
$$p(x) \rightarrow \exists x_1(p(x_1) \wedge \forall x_2(p(x_2) \vee r \rightarrow \neg p(x_2)))$$

Skolemisation: Choice Functions

We would like to get rid of existential quantifiers using choice functions, or witness functions.
Consider an example. We know that every tree has a root:

$$\forall x(\mathit{tree}(x) \rightarrow \exists y(\mathit{root}(y, x))). \qquad (*)$$

Then we can introduce a function, say *rootof* that gives the root of a tree and write

$$\forall x(\mathit{tree}(x) \rightarrow \mathit{root}(\mathit{rootof}(x), x)). \qquad (**)$$

Note that $(*)$ is a logical consequence of $(**)$.

Skolemisation: Choice Functions

We would like to get rid of existential quantifiers using choice functions, or witness functions.
Consider an example. We know that every tree has a root:

$$\forall x(\mathit{tree}(x) \rightarrow \exists y(\mathit{root}(y, x))). \qquad (*)$$

Then we can introduce a function, say *rootof* that gives the root of a tree and write

$$\forall x(\mathit{tree}(x) \rightarrow \mathit{root}(\mathit{rootof}(x), x)). \qquad (**)$$

Note that $(*)$ is a logical consequence of $(**)$.

Skolemisation

Let A be a closed rectified formula in NNF and $(\exists x)B$ be a subformula of A. Let $(\forall x_1), \ldots, (\forall x_n)$ be all universal quantifiers such that $(\exists x)B$ is in the scope of these quantifiers. Then:

1. remove $(\exists x)$ from A.
2. replace x everywhere in A by $f(x_1, \ldots, x_n)$, where f is a new function symbol.

Skolemisation does not preserve equivalence but preserves satisfiability.

CNF Transformation

Take a first-order formula F.

1. transform it into NNF;
2. rectify it;
3. skolemise it;
4. remove all universal quantifiers;
5. transform to CNF the same way as propositional formulas.

CNF Transformation

Universal closure of a formula A is a formula

$$(\forall x_1) \ldots (\forall x_n) A,$$

denoted by $\forall A$, where $x_1, \ldots, x_n$ are all free variables of A.

CNF transformation transforms a closed formula F into a set of clauses $C_1, \ldots, C_n$ such that F is satisfiable if and only if so is the set of formulas $\forall C_1, \ldots, \forall C_n$.

CNF Transformation

Universal closure of a formula A is a formula

$$(\forall x_1) \ldots (\forall x_n) A,$$

denoted by $\forall A$, where $x_1, \ldots, x_n$ are all free variables of A.

CNF transformation transforms a closed formula F into a set of clauses $C_1, \ldots, C_n$ such that F is satisfiable if and only if so is the set of formulas $\forall C_1, \ldots, \forall C_n$.

Outline

Example

Suppose we want to prove (establish validity of)

$$(\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y).$$

It is valid if and only if its negation

$$\neg((\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y))$$

is unsatisfiable.
The transformation of this formula to CNF gives us two clauses:

$$p(x,a)$$
$$\neg p(b,y).$$

Example

Suppose we want to prove (establish validity of)

$$(\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y).$$

It is valid if and only if its negation

$$\neg((\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y))$$

is unsatisfiable.

The transformation of this formula to CNF gives us two clauses:

$$p(x,a)$$
$$\neg p(b,y).$$

Example

Suppose we want to prove (establish validity of)

$$(\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y).$$

It is valid if and only if its negation

$$\neg((\exists y)(\forall x)p(x,y) \rightarrow (\forall x)(\exists y)p(x,y))$$

is unsatisfiable.
The transformation of this formula to CNF gives us two clauses:

$$p(x,a)$$
$$\neg p(b,y).$$

Example

How can we check unsatisfiability of

$$(\forall x)p(x, a)$$
$$(\forall y)\neg p(b, y)?$$

- Since we have $(\forall x)p(x, a)$, we also have $p(b, a)$;
- Since we have $(\forall y)\neg p(b, y)$, we also have $\neg p(b, a)$;
- $p(b, a)$ and $p(b, a)$ are unsatisfiable (e.g., by resolution).

Example

How can we check unsatisfiability of

$$(\forall x)p(x, a)$$
$$(\forall y)\neg p(b, y)?$$

- Since we have $(\forall x)p(x, a)$, we also have $p(b, a)$;
- Since we have $(\forall y)\neg p(b, y)$, we also have $\neg p(b, a)$;
- $p(b, a)$ and $p(b, a)$ are unsatisfiable (e.g., by resolution).

Example

How can we check unsatisfiability of

$$(\forall x)p(x, a)$$
$$(\forall y)\neg p(b, y)?$$

- Since we have $(\forall x)p(x, a)$, we also have $p(b, a)$;
- Since we have $(\forall y)\neg p(b, y)$, we also have $\neg p(b, a)$;
- $p(b, a)$ and $p(b, a)$ are unsatisfiable (e.g., by resolution).

Example

How can we check unsatisfiability of

$$(\forall x)p(x, a)$$
$$(\forall y)\neg p(b, y)?$$

- Since we have $(\forall x)p(x, a)$, we also have $p(b, a)$;
- Since we have $(\forall y)\neg p(b, y)$, we also have $\neg p(b, a)$;
- $p(b, a)$ and $p(b, a)$ are unsatisfiable (e.g., by resolution).

Ideas

Note that we established unsatisfiability by

- Substituting terms for variables, e.g. b for x in $p(x, a)$;
- Using propositional resolution.

Are these two ingredients sufficient to have a complete procedure?

Ideas

Note that we established unsatisfiability by

- Substituting terms for variables, e.g. b for x in $p(x, a)$;
- Using propositional resolution.

Are these two ingredients sufficient to have a complete procedure?

Substitution

- A substitution θ is a mapping from variables to terms such that the set $\{x \mid \theta(x) \neq x\}$ is finite.
- This set is called the domain of θ.
- Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are pairwise different variables, denotes the substitution θ such that

$$\theta(x) = \begin{cases} t_i & \text{if } x = x_i; \\ x & \text{if } x \notin \{x_1, \ldots, x_n\}. \end{cases}$$

- Application of this substitution to an expression E: simultaneous replacement of x_i by t_i.
- The result of the application of a substitution θ to E is denoted by $E\theta$.
- Since substitutions are functions, we can define their composition (writen $\sigma\tau$ instead of $\tau \circ \sigma$). Note that we have $E(\sigma\tau) = (E\sigma)\tau$.

Substitution

- A substitution θ is a mapping from variables to terms such that the set $\{x \mid \theta(x) \neq x\}$ is finite.
- This set is called the domain of θ.
- Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are pairwise different variables, denotes the substitution θ such that

$$\theta(x) = \begin{cases} t_i & \text{if } x = x_i; \\ x & \text{if } x \notin \{x_1, \ldots, x_n\}. \end{cases}$$

- Application of this substitution to an expression E: simultaneous replacement of x_i by t_i.
- The result of the application of a substitution θ to E is denoted by $E\theta$.
- Since substitutions are functions, we can define their composition (writen $\sigma\tau$ instead of $\tau \circ \sigma$). Note that we have $E(\sigma\tau) = (E\sigma)\tau$.

Substitution

- A substitution θ is a mapping from variables to terms such that the set $\{x \mid \theta(x) \neq x\}$ is finite.
- This set is called the domain of θ.
- Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are pairwise different variables, denotes the substitution θ such that

$$\theta(x) = \begin{cases} t_i & \text{if } x = x_i; \\ x & \text{if } x \notin \{x_1, \ldots, x_n\}. \end{cases}$$

- Application of this substitution to an expression E: simultaneous replacement of x_i by t_i.
- The result of the application of a substitution θ to E is denoted by $E\theta$.
- Since substitutions are functions, we can define their composition (writen $\sigma\tau$ instead of $\tau \circ \sigma$). Note that we have $E(\sigma\tau) = (E\sigma)\tau$.

Substitution

- A substitution θ is a mapping from variables to terms such that the set $\{x \mid \theta(x) \neq x\}$ is finite.
- This set is called the domain of θ.
- Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are pairwise different variables, denotes the substitution θ such that

$$\theta(x) = \begin{cases} t_i & \text{if } x = x_i; \\ x & \text{if } x \notin \{x_1, \ldots, x_n\}. \end{cases}$$

- Application of this substitution to an expression E: simultaneous replacement of x_i by t_i.
- The result of the application of a substitution θ to E is denoted by $E\theta$.
- Since substitutions are functions, we can define their composition (writen $\sigma\tau$ instead of $\tau \circ \sigma$). Note that we have $E(\sigma\tau) = (E\sigma)\tau$.

Substitution

- A substitution θ is a mapping from variables to terms such that the set $\{x \mid \theta(x) \neq x\}$ is finite.
- This set is called the domain of θ.
- Notation: $\{x_1 \mapsto t_1, \ldots, x_n \mapsto t_n\}$, where $x_1, \ldots, x_n$ are pairwise different variables, denotes the substitution θ such that

$$\theta(x) = \begin{cases} t_i & \text{if } x = x_i; \\ x & \text{if } x \notin \{x_1, \ldots, x_n\}. \end{cases}$$

- Application of this substitution to an expression E: simultaneous replacement of x_i by t_i.
- The result of the application of a substitution θ to E is denoted by $E\theta$.
- Since substitutions are functions, we can define their composition (writen $\sigma\tau$ instead of $\tau \circ \sigma$). Note that we have $E(\sigma\tau) = (E\sigma)\tau$.

Exercise

Suppose we have two substitutions

$$\{x_1 \mapsto s_1, \ldots, x_m \mapsto s_m\} \text{ and}$$
$$\{y_1 \mapsto t_1, \ldots, y_n \mapsto t_n\}.$$

How can we write their composition using the same notation?

Instance

An instance of an expression (that is term, atom, literal, or clause) E is obtained by applying a substitution to E. Examples:

- some instances of the term $f(x, a, g(x))$ are:
  $f(x, a, g(x))$,
  $f(y, a, g(y))$,
  $f(a, a, g(a))$,
  $f(g(b), a, g(g(b)))$;
- but the term $f(b, a, g(c))$ is not an instance of this term.

Ground instance: instance with no variables.

Instance

An instance of an expression (that is term, atom, literal, or clause) E is obtained by applying a substitution to E. Examples:

- some instances of the term $f(x, a, g(x))$ are:
  $f(x, a, g(x))$,
  $f(y, a, g(y))$,
  $f(a, a, g(a))$,
  $f(g(b), a, g(g(b)))$;
- but the term $f(b, a, g(c))$ is not an instance of this term.

Ground instance: instance with no variables.

Instance

An instance of an expression (that is term, atom, literal, or clause) E is obtained by applying a substitution to E. Examples:

- some instances of the term $f(x, a, g(x))$ are:
  $f(x, a, g(x))$,
  $f(y, a, g(y))$,
  $f(a, a, g(a))$,
  $f(g(b), a, g(g(b)))$;
- but the term $f(b, a, g(c))$ is not an instance of this term.

Ground instance: instance with no variables.

Instance

An instance of an expression (that is term, atom, literal, or clause) E is obtained by applying a substitution to E. Examples:

- some instances of the term $f(x, a, g(x))$ are:
  $f(x, a, g(x))$,
  $f(y, a, g(y))$,
  $f(a, a, g(a))$,
  $f(g(b), a, g(g(b)))$;
- but the term $f(b, a, g(c))$ is not an instance of this term.

Ground instance: instance with no variables.

Herbrand's Theorem

For a set of clauses S denote by S^* the set of ground instances of clauses in S.

Theorem (Herbrand)

Let S be a set of clauses. The following conditions are equivalent.

1. *S is unsatisfiable;*
2. *S^* is unsatisfiable;*

Note that by compactness the last condition is equivalent to

3. there exists a finite unsatisfiable set of ground instances of clauses in S.

The theorem reduces the problem of checking inconsistency of sets of arbitrary clauses to checking inconsistency of sets of ground clauses . . . the only problem is that S^* can be infinite even if S is finite.

Herbrand's Theorem

For a set of clauses S denote by S^* the set of ground instances of clauses in S.

Theorem (Herbrand)

Let S be a set of clauses. The following conditions are equivalent.

1. *S is unsatisfiable;*
2. *S^* is unsatisfiable;*

Note that by compactness the last condition is equivalent to

3. there exists a finite unsatisfiable set of ground instances of clauses in S.

The theorem reduces the problem of checking inconsistency of sets of arbitrary clauses to checking inconsistency of sets of ground clauses . . . the only problem is that S^* can be infinite even if S is finite.

Herbrand's Theorem

For a set of clauses S denote by S^* the set of ground instances of clauses in S.

Theorem (Herbrand)

Let S be a set of clauses. The following conditions are equivalent.

1. *S is unsatisfiable;*
2. *S^* is unsatisfiable;*

Note that by compactness the last condition is equivalent to

3. there exists a finite unsatisfiable set of ground instances of clauses in S.

The theorem reduces the problem of checking inconsistency of sets of arbitrary clauses to checking inconsistency of sets of ground clauses . . . the only problem is that S^* can be infinite even if S is finite.

Herbrand's Theorem

For a set of clauses S denote by S^* the set of ground instances of clauses in S.

Theorem (Herbrand)

Let S be a set of clauses. The following conditions are equivalent.

1. *S is unsatisfiable;*
2. *S^* is unsatisfiable;*

Note that by compactness the last condition is equivalent to

3. there exists a finite unsatisfiable set of ground instances of clauses in S.

The theorem reduces the problem of checking inconsistency of sets of arbitrary clauses to checking inconsistency of sets of ground clauses . . . the only problem is that S^* can be infinite even if S is finite.

Herbrand's Theorem

For a set of clauses S denote by S^* the set of ground instances of clauses in S.

Theorem (Herbrand)

Let S be a set of clauses. The following conditions are equivalent.

1. *S is unsatisfiable;*
2. *S^* is unsatisfiable;*

Note that by compactness the last condition is equivalent to

3. there exists a finite unsatisfiable set of ground instances of clauses in S.

The theorem reduces the problem of checking inconsistency of sets of arbitrary clauses to checking inconsistency of sets of ground clauses . . . the only problem is that S^* can be infinite even if S is finite.

Lifting

Lifting is a technique for proving completeness theorems in the following way:

1. Prove completeness of the system for a set of ground clauses;
2. Lift the proof to the non-ground case.

Lifting, Example

Consider two (non-ground) clauses $p(x, a) \vee q_1(x)$ and $\neg p(y, z) \vee q_2(y, z)$. If the signature contains function symbols, then both clauses have infinite sets of instances:

$$\begin{array}{rcl} \{p(r, a) \vee q_1(r) & \mid & r \text{ is ground}\} \\ \{\neg p(s, t) \vee q_2(s, t) & \mid & s, t \text{ are ground}\} \end{array}$$

We can resolve such instances if and only if $r = s$ and $t = a$. Then we can apply the following inference

$$\frac{p(s, a) \vee q_1(s) \quad \neg p(s, a) \vee q_2(s, a)}{q_1(s) \vee q_2(s, a)} \text{ (BR)}$$

But there is an infinite number of such inferences.

Lifting, Example

Consider two (non-ground) clauses $p(x, a) \vee q_1(x)$ and $\neg p(y, z) \vee q_2(y, z)$. If the signature contains function symbols, then both clauses have infinite sets of instances:

$$\begin{array}{rcl} \{p(r, a) \vee q_1(r) & \mid & r \text{ is ground}\} \\ \{\neg p(s, t) \vee q_2(s, t) & \mid & s, t \text{ are ground}\} \end{array}$$

We can resolve such instances if and only if $r = s$ and $t = a$. Then we can apply the following inference

$$\frac{p(s, a) \vee q_1(s) \quad \neg p(s, a) \vee q_2(s, a)}{q_1(s) \vee q_2(s, a)} \text{ (BR)}$$

But there is an infinite number of such inferences.

Lifting, Example

Consider two (non-ground) clauses $p(x, a) \vee q_1(x)$ and $\neg p(y, z) \vee q_2(y, z)$. If the signature contains function symbols, then both clauses have infinite sets of instances:

$$\begin{array}{rcl} \{p(r, a) \vee q_1(r) & \mid & r \text{ is ground}\} \\ \{\neg p(s, t) \vee q_2(s, t) & \mid & s, t \text{ are ground}\} \end{array}$$

We can resolve such instances if and only if $r = s$ and $t = a$. Then we can apply the following inference

$$\frac{p(s, a) \vee q_1(s) \quad \neg p(s, a) \vee q_2(s, a)}{q_1(s) \vee q_2(s, a)} \text{ (BR)}$$

But there is an infinite number of such inferences.

Lifting, Idea

The idea is to represent an infinite number of ground inferences of the form

$$\frac{p(s,a) \vee q_1(s) \quad \neg p(s,a) \vee q_2(s,a)}{q_1(s) \vee q_2(s,a)} \text{ (BR)}$$

by a single non-ground inference

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Is this always possible? Yes!

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Note that the substitution $\{x \mapsto y, z \mapsto a\}$ is a solution of the "equation" $p(x,a) = p(y,z)$.

Lifting, Idea

The idea is to represent an infinite number of ground inferences of the form

$$\frac{p(s,a) \vee q_1(s) \quad \neg p(s,a) \vee q_2(s,a)}{q_1(s) \vee q_2(s,a)} \text{ (BR)}$$

by a single non-ground inference

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Is this always possible? Yes!

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Note that the substitution $\{x \mapsto y, z \mapsto a\}$ is a solution of the "equation" $p(x,a) = p(y,z)$.

Lifting, Idea

The idea is to represent an infinite number of ground inferences of the form

$$\frac{p(s,a) \vee q_1(s) \quad \neg p(s,a) \vee q_2(s,a)}{q_1(s) \vee q_2(s,a)} \text{ (BR)}$$

by a single non-ground inference

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Is this always possible? Yes!

$$\frac{p(x,a) \vee q_1(x) \quad \neg p(y,z) \vee q_2(y,z)}{q_1(y) \vee q_2(y,a)} \text{ (BR)}$$

Note that the substitution $\{x \mapsto y, z \mapsto a\}$ is a solution of the "equation" $p(x,a) = p(y,z)$.

What should we lift?

- Selection function σ.
- Calculus $\mathbb{BR}_\sigma$.
- Ordering $\succ$, if we use ordered resolution.

Most importantly, for the lifting to work we should be able to solve equations $s \approx t$ between terms and between atoms.

What should we lift?

- Selection function σ.
- Calculus $\mathbb{BR}_\sigma$.
- Ordering $\succ$, if we use ordered resolution.

Most importantly, for the lifting to work we should be able to solve equations $s \simeq t$ between terms and between atoms.

Unifier

Unifier of expressions s_1 and s_2: a substitution θ such that $s_1\theta = s_2\theta$.

In other words, a unifier is a solution to an "equation" $s_1 = s_2$.

In a similar way we can define solutions to systems of equations $s_1 = s'_1, \ldots, s_n = s'_n$.
We call such solutions simultaneous unifiers of $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_n$.

Unifier

Unifier of expressions s_1 and s_2: a substitution θ such that $s_1\theta = s_2\theta$.

In other words, a unifier is a solution to an "equation" $s_1 = s_2$.

In a similar way we can define solutions to systems of equations $s_1 = s'_1, \ldots, s_n = s'_n$.

We call such solutions simultaneous unifiers of $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_n$.

Unifier

Unifier of expressions s_1 and s_2: a substitution θ such that $s_1\theta = s_2\theta$.

In other words, a unifier is a solution to an "equation" $s_1 = s_2$.

In a similar way we can define solutions to systems of equations $s_1 = s'_1, \ldots, s_n = s'_n$.
We call such solutions simultaneous unifiers of $s_1, \ldots, s_n$ and $s'_1, \ldots, s'_n$.

(Most General) Unifiers

A solution θ to a set of equations E is said to be a most general solution if for every other solution σ there exists a substitution τ such that $\theta\tau = \sigma$.
In a similar way can define a most general unifier.
Consider terms $f(x_1, g(x_1), x_2)$ and $f(y_1, y_2, y_2)$.
(Some of) their unifiers are
$\theta_1 = \{y_1 \mapsto x_1, y_2 \mapsto g(x_1), x_2 \mapsto g(x_1)\}$ and
$\theta_2 = \{y_1 \mapsto a, y_2 \mapsto g(a), x_2 \mapsto g(a), x_1 \mapsto a\}$:

$f(x_1, g(x_1), x_2)\theta_1 = f(x_1, g(x_1), g(x_1))$;
$f(y_1, y_2, y_2)\theta_1 = f(x_1, g(x_1), g(x_1))$;
$f(x_1, g(x_1), x_2)\theta_2 = f(a, g(a), g(a))$;
$f(y_1, y_2, y_2)\theta_2 = f(a, g(a), g(a))$.
But only θ_1 is most general.

Unification

Let E be a set of equations. An isolated equation in E is any equation $x = t$ in it such that x has exactly one occurrence in E.

input: a finite set of equations E
output: a solution to E or failure.
begin
  while there exists a non-isolated equation $(s = t) \in E$ **do**
    case (s, t) **of**
      $(t, t) \Rightarrow$ remove this equation from E
      $(x, t) \Rightarrow$ **if** x occurs in t **then** halt with failure
                else replace x by t in all other equations of E
      $(t, x) \Rightarrow$ replace this equation by $x = t$
                and do the same as in the case (x, t)
      $(c, d) \Rightarrow$ halt with failure
      $(c, f(t_1, \ldots, t_n)) \Rightarrow$ halt with failure
      $(f(t_1, \ldots, t_n), c) \Rightarrow$ halt with failure
      $(f(s_1, \ldots, s_m), g(t_1, \ldots, t_n)) \Rightarrow$ halt with failure
      $(f(s_1, \ldots, s_n), f(t_1, \ldots, t_n)) \Rightarrow$ replace this equation by the set
                $s_1 = t_1, \ldots, s_n = t_n$
  end while
  Now E has the form $\{x_1 = r_1, \ldots, x_l = r_l\}$ and every equation in it is isolated
  return the substitution $\{x_1 \mapsto r_1, \ldots, x_l \mapsto r_l\}$
end

Examples

$$\{h(g(f(x), a)) = h(g(y, y))\}$$
$$\{h(f(y), y, f(z)) = h(z, f(x), x)\}$$
$$\{h(g(f(x), z)) = h(g(y, y))\}$$

Occurs check

- The check "x occurs in t" is called an occurs check.
- In Prolog, the predicate $=$ implements unification without occurs check.
- There is also a predicate (and a command) for unification with occurs check.

Properties

Theorem Suppose we run the unification algorithm on $s = t$. Then

- If s and t are unifiable, then the algorithms terminates and outputs a most general unifier of s and t.
- If s and t are not unifiable, then the algorithms terminates with failure.

Notation (slightly ambiguous):

- $mgu(s, t)$ for a most general unifier;
- $mgs(E)$ for a most general solution.

Properties

Theorem Suppose we run the unification algorithm on $s = t$. Then

- If s and t are unifiable, then the algorithms terminates and outputs a most general unifier of s and t.
- If s and t are not unifiable, then the algorithms terminates with failure.

Notation (slightly ambiguous):

- $mgu(s, t)$ for a most general unifier;
- $mgs(E)$ for a most general solution.

Exercise

Consider a trivial system of equations $\{\}$ or $\{a = a\}$.

Which substitutions are solutions to it?

What is the set of most general solutions to it?

Exercise

Consider a trivial system of equations $\{\}$ or $\{a = a\}$.

Which substitutions are solutions to it?

What is the set of most general solutions to it?

Exercise

Consider a trivial system of equations $\{\}$ or $\{a = a\}$.

Which substitutions are solutions to it?

What is the set of most general solutions to it?

Properties

Theorem

Let C be a clause and E a set of equations. Then

$$\{D \in C^* \mid \exists\theta(C\theta = D \text{ and } \theta \text{ is a solution to } E)\} = (Cmgs(E))^*.$$

Outline

Binary Resolution System, Non-Ground Case

Binary resolution is the following inference rule:

$$\frac{\underline{A} \vee C \quad \underline{\neg B} \vee D}{(C \vee D)mgu(A, B)} \text{ (BR)},$$

Factoring is the following inference rule:

$$\frac{\underline{A} \vee \underline{B} \vee C}{(A \vee C)mgu(A, B)} \text{ (Fact)},$$

Soundness and Completeness

$\mathbb{BR}$ is sound and complete, that is, if a set of clauses is unsatisfiable, then one can derive an empty clause from this set.

Soundness is evident since the conclusion of any inference rule is a logical consequence of its premises.

Completeness can be proved using completeness of propositional resolution and lifting.

Soundness and Completeness

$\mathbb{BR}$ is sound and complete, that is, if a set of clauses is unsatisfiable, then one can derive an empty clause from this set.

Soundness is evident since the conclusion of any inference rule is a logical consequence of its premises.

Completeness can be proved using completeness of propositional resolution and lifting.

Soundness and Completeness

$\mathbb{BR}$ is sound and complete, that is, if a set of clauses is unsatisfiable, then one can derive an empty clause from this set.

Soundness is evident since the conclusion of any inference rule is a logical consequence of its premises.

Completeness can be proved using completeness of propositional resolution and lifting.

Ordered resolution?

Binary resolution with arbitrary selection is incomplete.

To define ordered resolution one has to define ordering for non-ground clauses in a way so that they also work for their ground instances.

Ordered resolution?

Binary resolution with arbitrary selection is incomplete.

To define ordered resolution one has to define ordering for non-ground clauses in a way so that they also work for their ground instances.

A problem

Is the following set of clauses unsatisfiable?

$$p(x, a)$$
$$\neg p(b, x)?$$

Yes, since clauses denote their universal closures:

$$(\forall x)p(x, a)$$
$$(\forall x)\neg p(b, x).$$

But no rule of the resolution system is applicable to these clauses.

A problem

Is the following set of clauses unsatisfiable?

$$p(x, a)$$
$$\neg p(b, x)?$$

Yes, since clauses denote their universal closures:

$$(\forall x)p(x, a)$$
$$(\forall x)\neg p(b, x).$$

But no rule of the resolution system is applicable to these clauses.

A problem

Is the following set of clauses unsatisfiable?

$$p(x, a)$$
$$\neg p(b, x)?$$

Yes, since clauses denote their universal closures:

$$(\forall x)p(x, a)$$
$$(\forall x)\neg p(b, x).$$

But no rule of the resolution system is applicable to these clauses.

Renaming away

The domain of a substitution θ is the set of variables $\{x \mid \theta(x) \neq x\}$ is finite.
The range of θ is the set of terms $\{x\theta \mid x\theta \neq x\}$.

A substitution θ is called renaming if (three equivalent characterisations)

- the domain of θ coincides with its range.
- θ has an inverse σ (that is, $\theta \circ \sigma = \sigma \circ \theta = \{\}$).
- there exists an n such that $\theta^n = \{\}$.

A variant of a term (atom, literal, clause) t is any term obtained from t by appying a renaming.

Renaming away

The domain of a substitution θ is the set of variables $\{x \mid \theta(x) \neq x\}$ is finite.
The range of θ is the set of terms $\{x\theta \mid x\theta \neq x\}$.

A substitution θ is called renaming if (three equivalent characterisations)

- the domain of θ coincides with its range.
- θ has an inverse σ (that is, $\theta \circ \sigma = \sigma \circ \theta = \{\}$).
- there exists an n such that $\theta^n = \{\}$.

A variant of a term (atom, literal, clause) t is any term obtained from t by appying a renaming.

Renaming away

The domain of a substitution θ is the set of variables $\{x \mid \theta(x) \neq x\}$ is finite.
The range of θ is the set of terms $\{x\theta \mid x\theta \neq x\}$.

A substitution θ is called renaming if (three equivalent characterisations)

- the domain of θ coincides with its range.
- θ has an inverse σ (that is, $\theta \circ \sigma = \sigma \circ \theta = \{\}$).
- there exists an n such that $\theta^n = \{\}$.

A variant of a term (atom, literal, clause) t is any term obtained from t by appying a renaming.

Hidden rule: renaming away

Renaming E_1 away from E_2: replace E_1 by its variant E_1' so that E_1' and E_2 have no common variables.

Before applying resolution to two clauses C_1 and C_2 we should always rename C_1 away from C_2.

Renaming is sometimes called standardising apart (especially in the logic programming literature).

Hidden rule: renaming away

Renaming E_1 away from E_2: replace E_1 by its variant E_1' so that E_1' and E_2 have no common variables.

Before applying resolution to two clauses C_1 and C_2 we should always rename C_1 away from C_2.

Renaming is sometimes called standardising apart (especially in the logic programming literature).

Hidden rule: renaming away

Renaming E_1 away from E_2: replace E_1 by its variant E_1' so that E_1' and E_2 have no common variables.

Before applying resolution to two clauses C_1 and C_2 we should always rename C_1 away from C_2.

Renaming is sometimes called standardising apart (especially in the logic programming literature).

Example

(1)	$\neg p(x) \vee \neg q(y)$	input	
(2)	$\neg p(x) \vee q(y)$	input	
(3)	$p(x) \vee \neg q(y)$	input	
(4)	$p(x) \vee q(y)$	input	
(5)	$\neg p(x) \vee \neg p(y)$	BR	(1,2)
(6)	$\neg p(x)$	Fact	(5)
(7)	$p(x) \vee p(y)$	BR	(3,4)
(8)	$p(x)$	Fact	(7)
(9)	$\square$	BR	(6,8)