Logical reasoning and programming

Introduction, propositional logic, and SAT

Karel Chvalovský

CIIRC CTU

Do you know this aquatint?

Verify a CPU

Assume that you are developing CPUs and you want to know that your new CPU does what it is supposed to do. For example, you want to verify the floating point unit in it.

How would you do that?

Verify a CPU

Assume that you are developing CPUs and you want to know that your new CPU does what it is supposed to do. For example, you want to verify the floating point unit in it.

How would you do that?

Of course, use formal methods!

Pentium FDIV bug

It was a bug in the floating point unit that affected early Intel Pentium processors and costed Intel $475 M (1995). The reason was that 5 cells were missing in a programmable logic array.

An elegant demonstration of the problem is that

$$4195835/3145727$$

should return

$$1.333820\ldots$$

but affected processors returned

$$1.333739\ldots$$

As a consequence, Intel improved investments in formal verification efforts. For example, various floating-point algorithms were formally verified in HOL Light. Maybe surprisingly, it requires a non-trivial mathematics.

Foundations of mathematics

Mathematicians faced similar problems in the 19th century, because mathematics became more abstract and hence more surprising and “paradoxical” results were obtained.

This started a rapid development of logic in the late 19th century and early 20th century and led to even more paradoxes. Many of them occurred in set theory.

Russell's paradox

Let $R = \{ x : x \notin x \}$, then $R \in R$ iff $R \notin R$.

The moral of this is that without solid formal foundations surprising problems can emerge. In reality, even in mathematics fully formal reasoning is hardly ever used.

Mistakes in mathematical proofs

Vladimir Voevodsky (1966–2017) was a very famous mathematician, a professor at the Institute for Advanced Studies and a Fields medalist, who later in his career became interested in formal methods.

> *In 1999–2000 . . . [I discovered] that the proof of a key lemma in my paper contained a mistake and that the lemma, as stated, could not be salvaged. . . . This story got me scared. Starting from 1993, multiple groups of mathematicians studied my paper at seminars and used it in their work and none of them noticed the mistake. And it clearly was not an accident. A technical argument by a trusted author, which is hard to check and looks similar to arguments known to be correct is hardly ever checked in detail.*
>
> *Voevodsky, The Institute Letter Summer 2014*

Formal methods in Amazon

Amazon experimented with TLA (Temporal Logic of Actions):

> *In industry, formal methods have a reputation for requiring a huge amount of training and effort to verify a tiny piece of relatively straightforward code, so the return on investment is justified only in safety-critical domains (such as medical systems and avionics). Our experience with TLA+ shows this perception to be wrong. At the time of this writing, Amazon engineers have used TLA+ on 10 large complex real-world systems. In each, TLA+ has added significant value, either finding subtle bugs we are sure we would not have found by other means, or giving us enough understanding and confidence to make aggressive performance optimizations without sacrificing correctness.*
>
> *We also initially avoid mentioning what TLA stands for, as doing so would give an incorrect impression of complexity.*

(Newcombe et al. 2015)

Software correctness in Amazon (2023)

Cook: *The thing we've seen is it's really by need. The storage team, for example, is able to be much more agile and be much more aggressive in the programs that they write because of the formal methods. They're able to write code that otherwise they might not want to deploy because they wouldn't be as confident about it, and they're deploying four times as fast. It was an investment in agility that's really paid off.*

Kröning: *There are actually a good number of stories wherein engineering teams didn't dare to roll out a particular feature or design revision or design variant that offers clear benefits — like being faster, using less power — because they just couldn't gain the confidence that it's actually right under all circumstances.*

source: Automated reasoning at Amazon: A conversation

What is a formal logic?

It studies inferences. Given a statement φ and a collection of statements Γ, the main problem is whether

$$\varphi \text{ follows logically from } \Gamma.$$

The word “formal” here means that only the logical forms of statements matter.

Example

Let statements in Γ describe the rules of chess and φ be “black can always draw”.

Declarative programming

Specify a problem and ask queries.

LOVELAND, DONALD W., Automated Theorem Proving: A Logical Basis., Amsterdam etc., North-Holland Publishing Company, 406 pp., Dfl. 100,— (Fundamental Studies in Computer Science).

I agreed to review this book in the hope that its study would be an instructive experience, and my highest expectations have been surpassed: the book contains a wealth of information (an average of two definitions per page being a modest estimate)...

Chapter 2 gives a very clear description of the Davis–Putnam Procedure for refutation, and a description of Robinson's Resolution Procedure, which I found somewhat less clear because its essential component "unification" remains very complicated.

E. W. Dijkstra Archive: EWD669 (October 1982)

LOVELAND, DONALD W., Automated Theorem Proving: A Logical Basis., Amsterdam etc., North-Holland Publishing Company, 406 pp., Dfl. 100,— (Fundamental Studies in Computer Science).

> *One curious aspect of this book deserves to be mentioned: although its subject matter is the (stylized) application of the first-order predicate calculus, 1) the author hardly uses it himself, and, 2) when he does, he does so clumsily.*
>
> *Ad 1. All his proofs are entirely verbal as if the first-order predicate calculus did not exist: proofs of several pages running text are not unusual at all. This is the more amazing as his prose is often obscure or wrong...*

Syntax and semantics in logic

Syntax

We describe our language and hence we define well-formed statements, called formulae. We want a mechanical calculus that describes how to derive (prove) formulae.

Semantics

We describe the meaning of formulae. The main notions are validity and semantic consequence.

We always want our syntax and semantics to be adequate:

correctness only valid formulae are derivable (provable),

completeness all valid formulae are derivable (provable).

Fragments from the history of logic

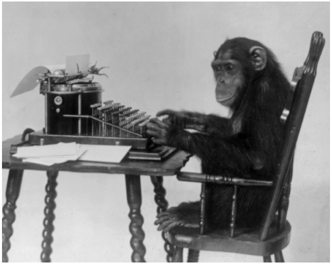
- Aristotle (384–322 BC) — syllogisms
- Gottfried Wilhelm Leibniz (1646–1716) — the first attempt to reduce a logical reasoning to a mechanical process
- George Boole (1815–1864) — Boolean logic
- Modern logic (Frege, . . .) was developed mainly to deal with issues in the foundations of mathematics in the late 19th century and early 20th century. Moreover, it is closely connected with the development of computers.

Automated theorem proving (ATP)

We have machines so use them to prove things for us. How to do that?

British Museum algorithm

Exhaustively check all possibilities one by one.



If monkeys are placed in front of typewriters and they type in a guaranteed random fashion, they will reproduce all the books of the library of the British Museum, provided they could type long enough.

(Wirth et al. 2009).

We usually have much better options. However, formal methods have their theoretical limits.

Limits of formal methods

Two famous fundamental theoretical problems are:

Incompleteness

Gödel's first incompleteness theorem says that it is impossible to describe basic arithmetic of the natural numbers by a set of axioms that is algorithmically recognizable.

Undecidability

Church and Turing famously proved that there is no decision procedure (algorithm) for validity in first-order logic (FOL). For example the halting problem is expressible in FOL.

It is good to be aware of these results, however, it is also good not to exaggerate them. We usually face more basic problems.

> *We may hope that machines will eventually compete with men in all purely intellectual fields.*

(Turing 1950, p. 460)

Practical limits of formal methods

We know from complexity theory many practically important decidable problems that are infeasible given current algorithms and computers, e.g., prime factorization.

Example (Find vs. verify a proof)

Is it possible to express 42 as a sum of three cubes of integers?

Practical limits of formal methods

We know from complexity theory many practically important decidable problems that are infeasible given current algorithms and computers, e.g., prime factorization.

Example (Find vs. verify a proof)

Is it possible to express 42 as a sum of three cubes of integers?

$$(-80538738812075974)^3 + 80435758145817515^3 + 12602123297335631^3 = 42$$

Found by Booker and Sutherland in 2019, check Sum of three cubes for other similar problems. Note that it is unknown whether the problem of sums of three cubes is decidable.

What are some areas where formal methods occur?

- model finding, planning, ...
  - conda package manager — the package data and constraints are expressed as a SAT problem, for details see Understanding and Improving Conda's performance (`mamba`→`libsolv`→SAT)
- verification
  - bug-finding in hardware, software, protocols
  - Microsoft, Intel, Amazon, Google, NASA, ...
  - software
    - seL4 — verified operating system microkernel
    - CompCert — verified C compiler
    - EURO-MILS — verified virtualization platform
    - CakeML — verified compiler for Standard ML
  - mathematics
    - the Kepler conjecture, Liquid Tensor Experiment
- mathematics
  - the Robbins problem
  - the Boolean Pythagorean triples problem

How to select a formal system?

We choose a formal system that is expressive enough to (reasonably) describe the problem and we usually prefer the weakest such system for computational reasons.

Examples of used formal systems include

- propositional logic — for problems in NP (or $\text{co-}\mathrm{NP}$),
- quantified Boolean formulae (QBF) — for problems in PSPACE,
- modal (temporal) logics — in verification,
- satisfiability modulo theories (SMT) — decidable problems,
- first-order logic (FOL),
- higher-order logics (HOL).

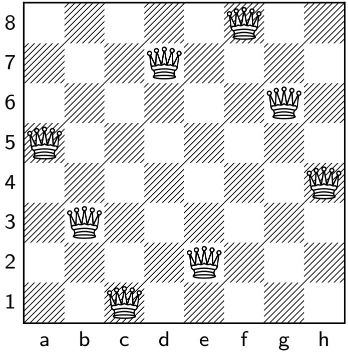
This course

We will introduce several systems with an “increasing” expressive power and then we sacrifice some expressive power for nice computational properties.

They can be very roughly divided into three groups:

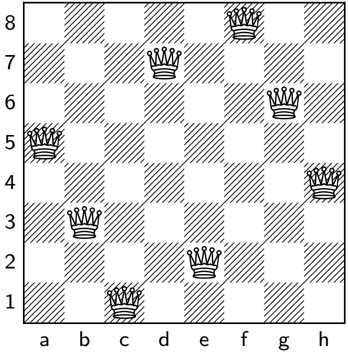
- propositional logic and its “extensions” — SAT and SMT,
- full first-order logic, and
- a “fragment” of first-order logic with nice computational properties — logic programming, Prolog, ASP, . . .

How should we represent the n-queens problem?



- We have squares and queens; it is sufficient to express facts about queens occupying particular squares...
- It is a Boolean problem—occupies/does not occupy, true/false, on/off, 1/0, ...
- We know that many useful things can be expressed by using "just" 1/0 and Boolean functions!

A possible representation of the n-queens problem



- We represent the fact that a queen occupies a square by propositional variables $q_{a1}, q_{b1}, \ldots, q_{h8}$.
- There should be rules expressing
  - n queens on the board (or at least one per row or column),
  - no two queens on the same row,
  - no two queens on the same column,
  - no two queens on the same diagonal.

Propositional logic (recap)

Simple, yet quite powerful, formal system. We have elementary propositions called atomic formulae, or atoms, which can be assigned *truth values*[1], and combine them using Boolean connectives (functions) into more complex propositions.

Example

If it rains, then I stay at home.

- "it rains" and "I stay at home" are propositions, say p and q, respectively.
- "if ..., then ..." is a connective, called implication and denoted $\rightarrow$.
- Hence the logical form of the sentence in propositional logic is $p \rightarrow q$.

[1] 1 is true and 0 is false.

Formulae

Placeholders for atomic formulae (propositions) are called propositional variables Var, say $p, q, r, \ldots$. We also have a unary connective negation ($\neg$) and binary connectives conjunction ($\wedge$), disjunction ($\vee$), and implication ($\rightarrow$).[2]

Definition

The set of propositional formulae Fml is the smallest set satisfying:

- every propositional variable from Var is a formula,
- if φ is a formula, then $(\neg\varphi)$ is a formula,
- if φ and ψ are formulae, then $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, and $(\varphi \rightarrow \psi)$ are formulae.

We usually write only parentheses that are necessary for unambiguous reading.

[2] We also use $\varphi \leftrightarrow \psi$ as a shortcut for $(\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi)$.

The meaning of formulae

Formulae are Boolean functions and hence we can use truth tables

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

and

p	q	$\neg p \vee q$
0	0	1
0	1	1
1	0	0
1	1	1

To be equivalent means to have the same truth tables.

The meaning of formulae

Formulae are Boolean functions and hence we can use truth tables

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

and

p	q	$\neg p \vee q$
0	0	1
0	1	1
1	0	0
1	1	1

To be equivalent means to have the same truth tables. Really?

p	q	r	$(p \to q) \wedge (r \to r)$	$p \to q$
0	0	0	1	1
0	0	1	1	1
0	1	0	1	1
0	1	1	1	1
1	0	0	0	0
1	0	1	0	0
1	1	0	1	1
1	1	1	1	1

Semantics

We formally describe the meaning of formulae.

A valuation v is an assignment of truth values to propositional variables, that is a function $v\colon \mathit{Var} \to \{0,1\}$. It can be uniquely extended to all formulae, because connectives are functions of truth values, and we freely use valuations this way.

Hence $v(\neg\varphi) = 1 - v(\varphi)$ and $v(\varphi \circ \psi) = v(\varphi) \bullet v(\psi)$, for $\circ \in \{\wedge, \vee, \to\}$, where $\bullet$ is the Boolean function defining $\circ$.

If $v(\varphi) = 1$, then we also write $v \models \varphi$ and say "formula φ is satisfied by valuation v" or "valuation v satisfies formula φ".

Example

If $v(p) = 1$ and $v(q) = 0$, then $v(p \to q) = 0$ and $v(p \vee q) = 1$.

Truth tables

The following truth table describes four valuations

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

Remark

Let φ be a formula and v, v' be two valuations such that they are equal on all propositional variables occurring in φ, then clearly $v(\varphi) = v'(\varphi)$. Hence only the valuation of variables occurring in a formula matters.

Truth tables

The following truth table describes four valuations, or strictly speaking infinitely many of them...

p	q	$p \to q$
0	0	1
0	1	1
1	0	0
1	1	1

p	q	r	$\cdots$	$p \to q$
0	0	0		1
0	0	1		1
0	1	0		1
0	1	1	$\cdots$	1
1	0	0		0
1	0	1		0
1	1	0		1
1	1	1		1
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$

Remark

Let φ be a formula and v, v' be two valuations such that they are equal on all propositional variables occurring in φ, then clearly $v(\varphi) = v'(\varphi)$. Hence only the valuation of variables occurring in a formula matters.

Semantic consequence I.

A formula φ *follows from* (or *is a consequence of*) *a formula* ψ, we write $\psi \models \varphi$, if φ is satisfied by every valuation v that satisfies ψ.

Example

$p \wedge q \models p$ and $p \models q \to q$.

Relation $\models$ is clearly reflexive and transitive, but not symmetric.

Two formulae φ and ψ are equivalent, we write $\varphi \equiv \psi$ or $\varphi \models\mid \psi$, if $\varphi \models \psi$ and $\psi \models \varphi$.

A very important property of propositional logic is that we can freely replace a subformula[3] by an equivalent formula. Formally, let ψ be a subformula of φ and $\psi \equiv \chi$. If we replace ψ in φ by χ, then the resulting formula is equivalent to φ.

Example

$(p \to q) \equiv (\neg p \vee q)$ and hence $((p \to q) \vee r) \equiv ((\neg p \vee q) \vee r)$.

[3] A formula ψ is a *subformula* of a formula φ if ψ is a substring of φ.

Some useful properties of $\{\neg, \wedge, \vee\}$

The following equivalences hold

- $\varphi \equiv \neg\neg\varphi$, (double negation)
- $\varphi \equiv \varphi \circ \varphi$, for $\circ \in \{\wedge, \vee\}$, (idempotency)
- $\varphi \circ \psi \equiv \psi \circ \varphi$, for $\circ \in \{\wedge, \vee\}$, (commutativity)
- $\varphi \circ (\psi \circ \chi) \equiv (\varphi \circ \psi) \circ \chi$ for $\circ \in \{\wedge, \vee\}$, (associativity)
- $\neg(\varphi \wedge \psi) \equiv \neg\varphi \vee \neg\psi$, (DeMorgan's law)
- $\neg(\varphi \vee \psi) \equiv \neg\varphi \wedge \neg\psi$, (DeMorgan's law)
- $\varphi \wedge (\psi \vee \chi) \equiv (\varphi \wedge \psi) \vee (\varphi \wedge \chi)$, (distributivity)
- $\varphi \vee (\psi \wedge \chi) \equiv (\varphi \vee \psi) \wedge (\varphi \vee \chi)$ (distributivity)

for all formulae φ, ψ, and χ.

Thanks to associativity we can write $\varphi_1 \circ \cdots \circ \varphi_n$ without parentheses, for $\circ \in \{\wedge, \vee\}$.

Some useful properties of $\to$

The following equivalences hold

- $\varphi \to \psi \equiv \neg\varphi \vee \psi$,
- $\varphi \to \psi \equiv \neg(\varphi \wedge \neg\psi)$,
- $\varphi \to \psi \equiv \neg\psi \to \neg\varphi$,
- $\varphi \to (\psi \to \chi) \equiv \psi \to (\varphi \to \chi)$,
- $\varphi \to (\psi \to \chi) \equiv (\varphi \wedge \psi) \to \chi$,
- $(\varphi_1 \wedge \cdots \wedge \varphi_n) \to (\psi_1 \vee \cdots \vee \psi_m) \equiv \neg\varphi_1 \vee \cdots \vee \neg\varphi_n \vee \psi_1 \vee \cdots \vee \psi_m$

for all formulae φ, ψ, χ, φ_i, and ψ_i.

Semantic consequence II.

A formula φ *follows from a set of formulae* Γ, we write $\Gamma \models \varphi$, if φ is satisfied by every valuation v that satisfies all formulae in Γ.

$$\Gamma \models \varphi \text{ iff } \forall v (\text{if } v \models \Gamma \text{ then } v \models \varphi),$$

where $v \models \Gamma$ means that $v \models \psi$ for all $\psi \in \Gamma$. We also say that φ is a consequence of Γ.

The relation is clearly monotone; if $\Gamma \models \varphi$, then $\Gamma \cup \Delta \models \varphi$.

We have (semantic) deduction theorem $\Gamma \cup \varphi \models \psi$ iff $\Gamma \models \varphi \to \psi$. Hence $\varphi \models \psi$ iff $\models \varphi \to \psi$. Hence $\varphi \equiv \psi$ iff $\models \varphi \leftrightarrow \psi$.

Example

$$\begin{aligned} p, p \to q, q \to r &\models r \\ p \to q, q \to r &\models p \to r \\ p \to q &\models (q \to r) \to (p \to r) \\ &\models (p \to q) \to ((q \to r) \to (p \to r)) \end{aligned}$$

Satisfiable formulae and tautologies

We say that a formula φ is

- *satisfiable* if there is a valuation v s.t. $v(\varphi) = 1$, that is $v \models \varphi$,
- *tautology* if for every valuation v holds $v(\varphi) = 1$, that is $\models \varphi$,
- *contradiction* if for every valuation v holds $v(\varphi) = 0$, we also call it *unsatisfiable*.

We call the set of all satisfiable and tautological formulae $\mathtt{SAT}$ and $\mathtt{TAUT}$, respectively.

Example

$p \wedge \neg q \in \mathtt{SAT}$, $p \vee \neg p \in \mathtt{TAUT}$, and $p \wedge \neg p \notin \mathtt{SAT}$.

For any formula φ and a finite set of formulae Γ we have

$$\varphi \in \mathtt{TAUT} \quad \text{iff} \quad \neg\varphi \notin \mathtt{SAT}.$$

Hence

$$\Gamma \models \varphi \quad \text{iff} \quad \bigwedge \Gamma \to \varphi \in \mathtt{TAUT} \quad \text{iff} \quad \bigwedge \Gamma \wedge \neg\varphi \notin \mathtt{SAT}.$$

Some further useful properties

It is possible to have both $\varphi \in \mathtt{SAT}$ and $\neg\varphi \in \mathtt{SAT}$.

If $\varphi \in \mathtt{TAUT}$, then $\varphi \in \mathtt{SAT}$. Hence $\mathtt{TAUT} \subset \mathtt{SAT}$.

A set of formulae Γ is satisfiable, we write $\Gamma \in \mathtt{SAT}$, if there is a valuation v such that $v \models \varphi$ for every formula $\varphi \in \Gamma$.

If $\Gamma \cup \Delta \in \mathtt{SAT}$, then $\Gamma \in \mathtt{SAT}$ and $\Delta \in \mathtt{SAT}$.

It is known that deciding $\varphi \in \mathtt{SAT}$ is an NP-complete problem and hence $\varphi \in \mathtt{TAUT}$ is a $\textsc{co}\text{-}\mathrm{NP}$-complete problem. Therefore any problem in NP can be formulated as a satisfiability question, without greatly (see polynomial reductions) increasing the problem size.

Special formulae $\top$ and $\bot$

We either define special formulae $\top$ and $\bot$ directly as propositional constants (nullary connectives), $v(\top) = 1$ and $v(\bot) = 0$ for every valuation v, or equivalently we can define them as shortcuts $\top = p \vee \neg p$ and $\bot = p \wedge \neg p$.

The following relations hold

- $\models \top$,
- $\bot \models \varphi$,
- if $\varphi \in \mathtt{TAUT}$, then $\varphi \equiv \top$,
- if $\varphi \notin \mathtt{SAT}$, then $\varphi \equiv \bot$

for every formula φ.

Various representations of Boolean functions

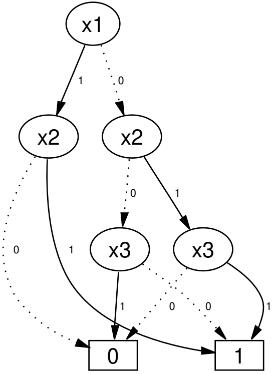
A formula is just one possible representation of a Boolean function.

There are various ways how to represent Boolean functions

- truth tables,
- binary decision trees,
- binary decision diagrams,
- ...

Of course, the best representation depends on what we want to do with it.

There is a popular way how to express propositional formulae that fits our purposes.



source: BDD (wiki)

Normal forms

A *literal* is a propositional variable p (positive l.) or a negation of propositional variable $\neg p$ (negative l.). In this context we write $\overline{p}$ instead of $\neg p$.

A *clause* is any disjunction of finitely many literals. An important special case is the empty clause, we write $\square$.

A formula φ is in *conjunctive normal form* (CNF) if φ is a conjunction of clauses. Analogously *disjunctive normal form* (DNF) is defined as a disjunction of conjunctions of literals.

Remark

The clause

$$\overline{p_1} \vee \overline{p_2} \vee \cdots \vee \overline{p_m} \vee q_1 \vee q_2 \vee \cdots \vee q_n$$

is equivalent (for example) to

$$(p_1 \wedge p_2 \wedge \cdots \wedge p_m) \to (q_1 \vee q_2 \vee \cdots \vee q_n).$$

Why are CNFs and DNFs important?

Theorem

For every formula φ exists formulae φ' in CNF and φ'' in DNF such that φ, φ', and φ'' are all equivalent.

Example

Formula

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

is equivalent to

$$(\overline{p} \vee q) \wedge (\overline{q} \vee p)$$

and

$$(\overline{p} \wedge \overline{q}) \vee (p \wedge q).$$

How to obtain DNFs and CNFs?

It is easy to obtain an equivalent DNF formula using truth tables:

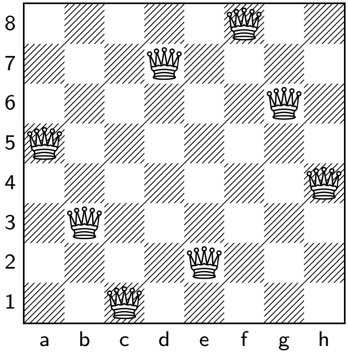
p	q	$(p \to q) \land (q \to p)$	
0	0	1	$\bar{p} \land \bar{q}$
0	1	0	
1	0	0	
1	1	1	$p \land q$

A formula in DNF obtained this way is in so called full disjunctive normal form. It is a unique representation up to ordering.

Analogously, we can produce a formula in CNF. How?

However, this is usually a very inefficient approach.

Satisfiability



- It is common to express problems in terms of satisfiability.
- We want to simultaneously satisfy
  - n queens on the board (or at least one per row or column) and
  - no two queens on the same row and
  - no two queens on the same column and
  - no two queens on the same diagonal.

Satisfiability and normal forms

It is easy to test whether a formula in DNF

$$(\cdots \wedge \cdots \wedge \cdots) \vee \cdots \vee (\cdots \wedge \cdots \wedge \cdots)$$

is satisfiable, but transforming a formula into DNF can lead to an exponential increase in the size of formula, see later. Hence we, perhaps surprisingly, prefer CNF

$$(\cdots \vee \cdots \vee \cdots) \wedge \cdots \wedge (\cdots \vee \cdots \vee \cdots)$$

for testing satisfiability. The reasons will be clear later on.

Moreover, it is very convenient to deal with CNFs if a problem is expressible as a conjunction of properties.

CNFs by rewriting

We obtain an equivalent formula in CNF using the following steps:

1. First, use the following rewriting rules as long as possible and obtain so-called negation normal form (NNF):

$$\begin{array}{rcll} \varphi \to \psi & \rightsquigarrow & \neg\varphi \vee \psi & \\ \neg\neg\varphi & \rightsquigarrow & \varphi & \\ \neg(\varphi \vee \psi) & \rightsquigarrow & \neg\varphi \wedge \neg\psi & \text{DeMorgan's law} \\ \neg(\varphi \wedge \psi) & \rightsquigarrow & \neg\varphi \vee \neg\psi & \text{DeMorgan's law} \end{array}$$

2. Second, distribute disjunctions until CNF is obtained:

$$\begin{array}{rcl} \varphi \vee (\psi \wedge \chi) & \rightsquigarrow & (\varphi \vee \psi) \wedge (\varphi \vee \chi) \\ (\psi \wedge \chi) \vee \varphi & \rightsquigarrow & (\psi \vee \varphi) \wedge (\chi \vee \varphi) \end{array}$$

Some properties of normal forms

Formulae can be transformed to normal forms in many ways and this can significantly influence their size and also the behavior of algorithms used for testing satisfiability.

Uniqueness

Normal forms are not necessarily unique, e.g.,

$$(p \rightarrow q) \wedge (q \rightarrow r) \wedge (r \rightarrow p)$$

is equivalent to both

$$(\overline{p} \vee q) \wedge (\overline{q} \vee r) \wedge (\overline{r} \vee p) \quad \text{and} \quad (\overline{p} \vee r) \wedge (\overline{q} \vee p) \wedge (\overline{r} \vee q).$$

Equivalent normal forms can be exponentially longer

Transforming

$$\varphi = (p_1 \wedge q_1) \vee \cdots \vee (p_n \wedge q_n)$$

into CNF leads to

$$\varphi' = \bigwedge_{r_1,\ldots,r_n} r_1 \vee r_2 \vee \cdots \vee r_n, \text{ where either } r_i = p_i, \text{ or } r_i = q_i.$$

Hence the length of φ' is $\mathcal{O}(2^n)$, but the length of φ is $\mathcal{O}(n)$.

Another important example, the n-variable parity function (equivalent to $x_1 \oplus x_2 \oplus \cdots \oplus x_n$) has only long ($\mathcal{O}(2^{n-1})$) CNF and DNF representations.

Tseytin transformation

We can avoid this possible exponential blowup[4] by introducing new variables that encode values of all subformulae in the original formula. The obtained formula is not equivalent (it has new variables) to the original one, but they are *equisatisfiable*—either both formulae are satisfiable, or both are unsatisfiable.

Example

For $\varphi = (p_1 \wedge q_1) \vee \cdots \vee (p_n \wedge q_n)$ we set

$$r_i \leftrightarrow (p_i \wedge q_i),$$

for $1 \leq i \leq n$, that is equivalent to

$$(\overline{p_i} \vee \overline{q_i} \vee r_i) \wedge (p_i \vee \overline{r_i}) \wedge (q_i \vee \overline{r_i}).$$

Taking a conjunction of all these formulae and $r_1 \vee \cdots \vee r_n$ gives us a formula φ' in CNF such that φ and φ' are equisatisfiable. Moreover, $|\varphi'| = \mathcal{O}(|\varphi|)$.

[4] If connectives occurring in the formula have linear clausal encodings.

Tseytin transformation — algorithm

Let φ be a non-atomic formula and $\psi_1, \ldots, \psi_m$ be all unique non-atomic subformulae of φ such that no ψ_i is a subformula of ψ_j if $1 \le j < i \le m$. Hence $\psi_m = \varphi$. Let $\{r_1, \ldots, r_m\}$ be fresh variables not occurring in φ.

Start with $\Delta = \emptyset$. Iteratively process ψ_i, for $1 \le i \le m$, as follows

if $\psi_i = \overline{p}$	add $(\overline{p} \vee \overline{r_i}) \wedge (p \vee r_i)$ to Δ,
if $\psi_i = p \wedge q$	add $(\overline{p} \vee \overline{q} \vee r_i) \wedge (p \vee \overline{r_i}) \wedge (q \vee \overline{r_i})$ to Δ,
if $\psi_i = p \vee q$	add $(p \vee q \vee \overline{r_i}) \wedge (\overline{p} \vee r_i) \wedge (\overline{q} \vee r_i)$ to Δ,
if $\psi_i = p \rightarrow q$	add $(\overline{p} \vee q \vee \overline{r_i}) \wedge (p \vee r_i) \wedge (\overline{q} \vee r_i)$ to Δ

and replace all occurrences of ψ_i in $\psi_{i+1}, \ldots, \psi_m$ by r_i.

The formulae $r_m \wedge \bigwedge \Delta$ and φ are equisatisfiable.

The algorithm can be improved, e.g., there is no need to encode negative literals, or one-sided variants.

SAT problem

Given a formula φ in CNF decide whether $\varphi \in \mathtt{SAT}$.

We can use truth tables, but that is in many cases too complicated. It means to test all possible valuations and for example

$$p \wedge \overline{p} \wedge (q_1 \vee \cdots \vee q_n)$$

is clearly unsatisfiable regardless of values of $q_1, \ldots, q_n$.

In the next lecture we will present better ways how to test satisfiability. We can think about transformations of formulae that preserve satisfiability; a trivial example is to handle clauses as sets of literals and formulae in CNF as sets of clauses.

The Boolean Pythagorean triples problem

It was a long-standing open problem in Ramsey theory that was solved using a SAT solver in 2016. The proof requires 200TB (compressed 68GB) and was computed on a cluster with 800 cores in 2 days.

Positive integers a, b, c form a Pythagorean triple if $a^2 + b^2 = c^2$. We know, e.g., $3^2 + 4^2 = 5^2$.

Can the set $\{1, 2, 3 \dots\}$ of the positive integers be divided into two parts in such a way that no part contains a Pythagorean triple?

The set $\{1, \dots, 7824\}$ can be divided into two such parts, but that is not possible for $\{1, \dots, 7825\}$.

Note that there are 2^{7825} possible divisions in the later case and all these divisions must be ruled out. Hence some “clever” reasoning had to be used. For details see (Heule, Kullmann, and Marek 2016).

Bibliography I

Heule, Marijn J. H., Oliver Kullmann, and Victor W. Marek (2016). "Solving and Verifying the Boolean Pythagorean Triples Problem via Cube-and-Conquer". In: *Theory and Applications of Satisfiability Testing – SAT 2016: 19th International Conference, Bordeaux, France, July 5-8, 2016, Proceedings*. Ed. by Nadia Creignou and Daniel Le Berre. Cham: Springer International Publishing, pp. 228–245. ISBN: 978-3-319-40970-2. DOI: 10.1007/978-3-319-40970-2_15.

Newcombe, Chris et al. (Mar. 2015). "How Amazon Web Services Uses Formal Methods". In: *Communications of the ACM* 58.4, pp. 66–73. ISSN: 0001-0782. DOI: 10.1145/2699417.

Turing, Alan Mathison (Oct. 1950). "Computing machinery and intelligence". In: *Mind* LIX.236, pp. 433–460. ISSN: 0026-4423. DOI: 10.1093/mind/LIX.236.433.

Bibliography II

Wirth, Claus-Peter et al. (2009). "Jacques Herbrand: Life, Logic, and Automated Deduction". In: *Handbook of the History of Logic*. Elsevier, pp. 195–254. DOI: 10.1016/s1874-5857(09)70009-3.