Formale Methoden der Informatik

Block 2: Satisfiability Problems

1. Preparatory Concepts

Uwe Egly

Knowledge-Based Systems Group
Institute of Information Systems
Vienna University of Technology

General goal of Block 2 (SAT)

Provide the necessary tools and background info to construct a decision procedure for equality logic with uninterpreted functions (EUF), which has various application in hardware and software verification, e.g.:

- proving equivalence of two hardware designs, or
- proving correctness of a compiler by checking equivalence between the source and the target program.

Other decision procedures (related to verification) are the ones for linear arithmetic, bit vectors, arrays, pointer arithmetic, ...

General goal of Block 2 (cont'd)

Overall procedure

The problem of deciding a EUF-formula φ^{EUF} is stepwisely reduced to the SAT problem of a propositional formula φ^{P}, such that (s.t.)

$$\varphi^{EUF} \text{ is valid} \quad \text{iff} \quad \varphi^{P} \text{ is unsatisfiable}$$

Models of φ^{P} provide counterexamples for the validity of φ^{EUF}. As we will see, the reduction is PTIME-computable.

General goal of Block 2 (cont'd)

The main steps of the procedure

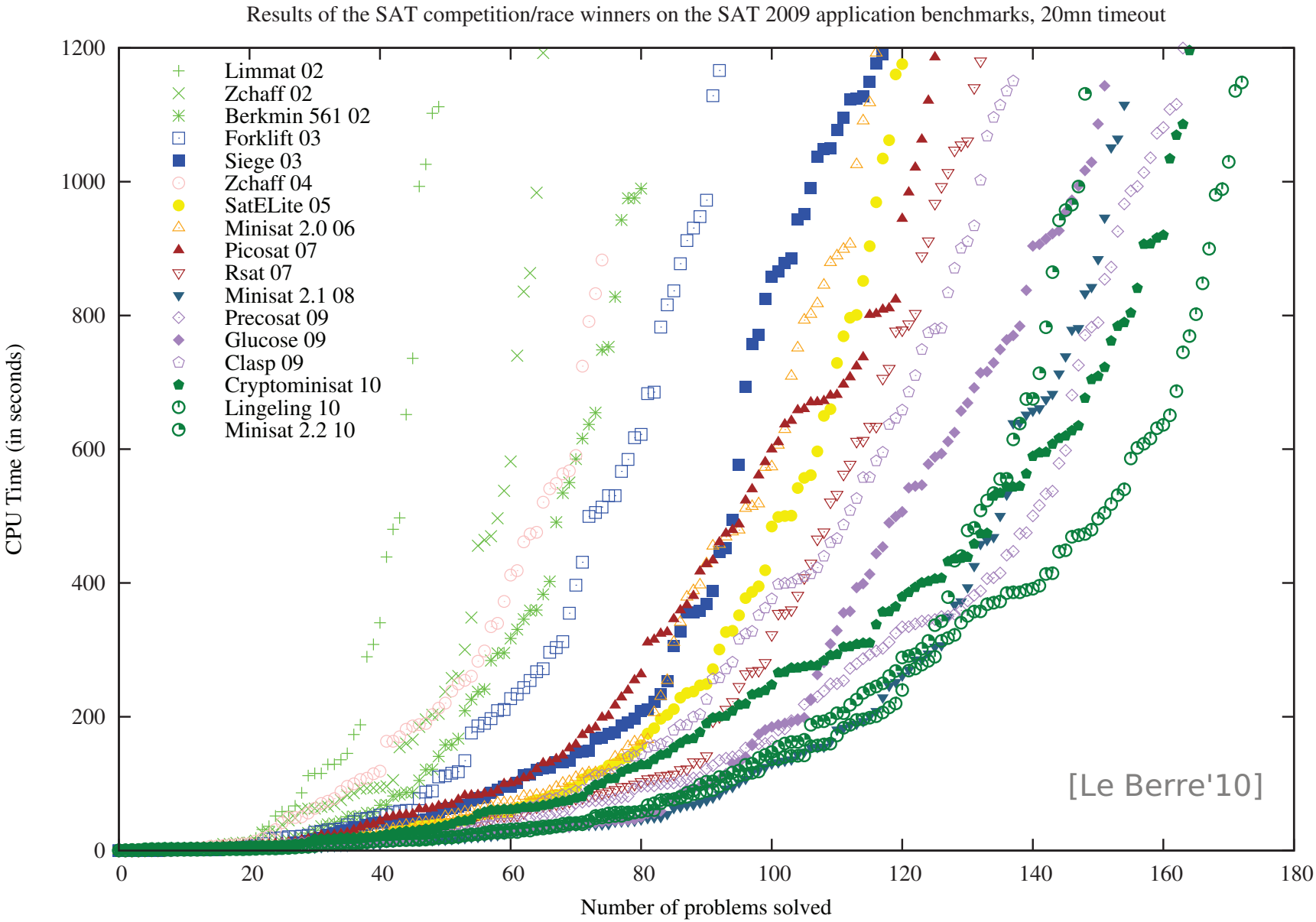
1. Reduce φ^{EUF} to a validity-equivalent formula φ^{E} with equality but without function symbols
2. Reduce φ^{E} to φ^{P} such that φ^{E} is valid iff φ^{P} is unsatisfiable

Q: Why do we use SAT as a target formalism for the reduction?

☞ Because of huge improvements of SAT solvers over the last decade!

Results of the SAT 2009 application benchmarks

for leading solvers from 2002 to 2010



The topics of this block

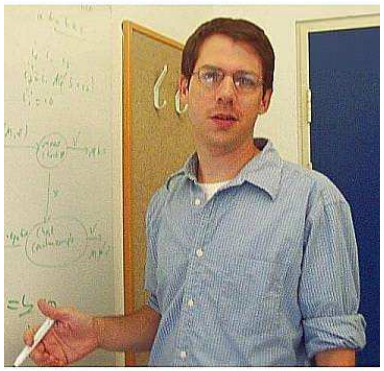
Or what do we need in order to achieve our goal?

1. Preparatory concepts
2. Techniques for modern SAT solvers
3. First-order theories
4. Equality logic
5. Equality logic and uninterpreted function symbols

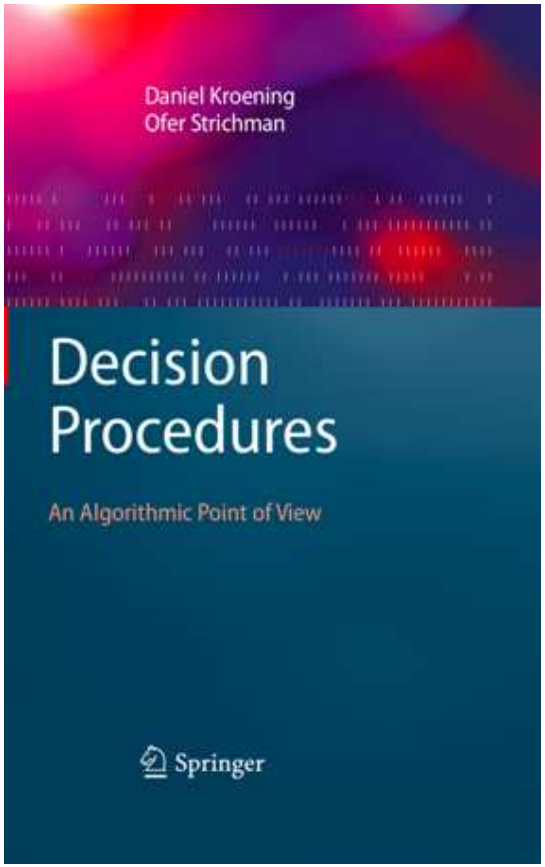
Background reading

Decision Procedures–an Algorithmic Point of View, Springer, 2008

Daniel Kroening



Ofer Strichman

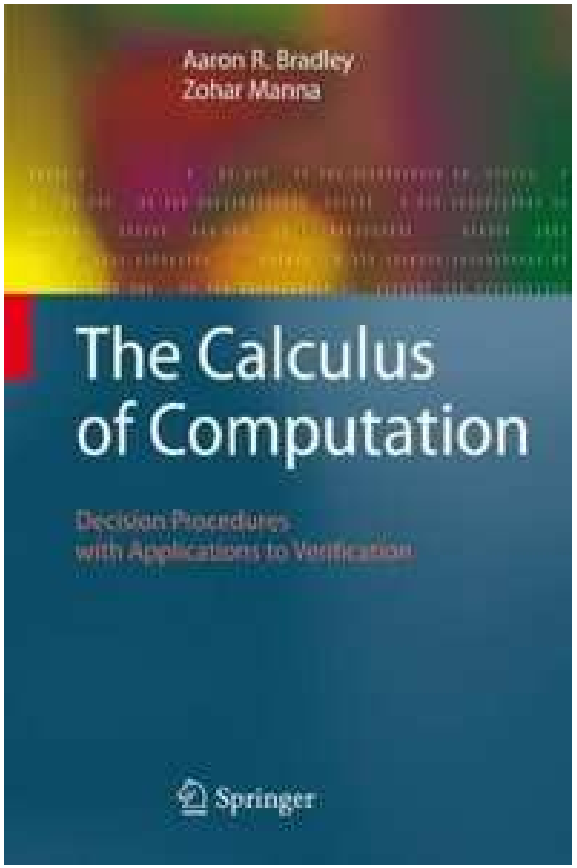


Background reading

The Calculus of Computation, Springer, 2007

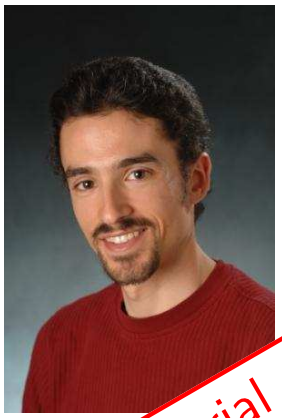
Aaron R. Bradley

Zohar Manna



Background reading

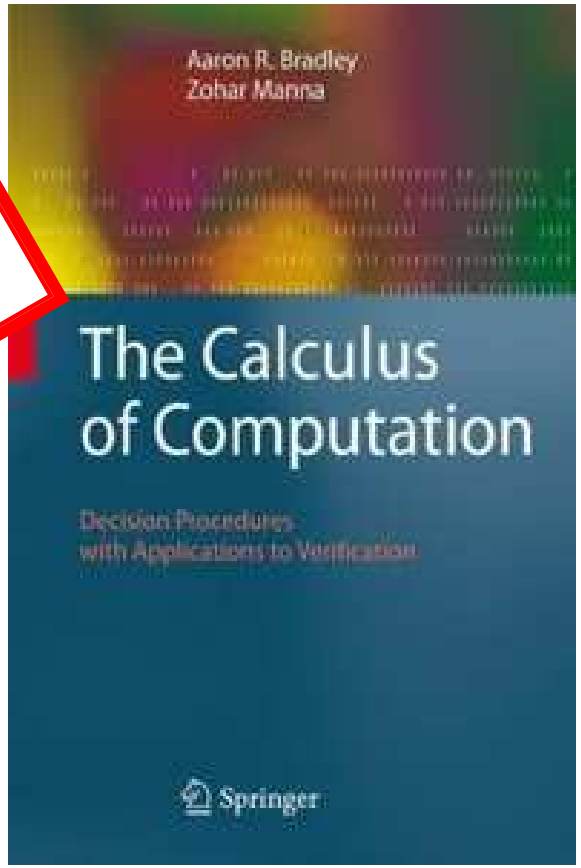
The Calculus of Computation, Springer, 2007



Aaron

Zohar Manna

Further material available for free!
See TUWEL for links to further (downloadable) material including articles and a PDP document relating the material to different topics.



Connections to other blocks of this lecture

From Block 1, we get

- the definition of the SAT problem and
- the principle of faithful PTIME computable reductions.

For Block 4, we provide

- the basics of SAT solving for, e.g., bounded model checking, and
- a fully worked-out example of a reduction of an interesting theory to SAT.

Outline

General information on Block 2

Introduction
- Motivation
- Examples
  - Equivalence checking of if-then-else statements
  - The circuit example

Syntax of propositional logic

Semantics of propositional logic
- Notations
- Entailment

Different normal forms and translation procedures

Learning Objectives

Why logic and formal methods in CS studies?

- Programming
  PROLOG, Boolean expressions in if-then, assertions, ...
- Languages for and reasoning about specifications,
  dynamic behavior (temporal logics), consistency, ...
- Reasoning about properties of SW, e.g., dependencies
  between packages
  http://old-de.opensuse.org/Paketverwaltung/SAT_Solver/Grundlagen
  http://files.opensuse.org/opensuse/en/b/b9/Fosdem2008-solver.pdf
- Knowledge representation
  Web ontologies, description logics, reasoning under
  incomplete/uncertain information, ...
- Hardware
  circuit specification, (automated) verification of circuits, ...
- and much more!!!

Why logic and formal methods in CS studies? (cont'd)

- Systems engineering for HW/SW is extremely error-prone
- There are many examples for disasters like the Ariane 5 Flight 501 which crashed due to an integer overflow, or the launch failure of a Mariner 1 rocket due to a missing hyphen. Consider the two links for more info.
  - [link to a wiki article on software engineering disasters]
  - [link to a wiki article on programming bugs]
- There is a strong demand to improve the product quality of both software and hardware (especially in safety-critical applications like embedded systems)

☞ Formal methods allow to define (1) what you want (spec) and (2) what you have (implementation). You can
  - verify that (1) and (2) are "identical" or
  - synthesize (2) from (1)

Types of logics interesting for CS

There is not *the* logic for CS, but many of them

- Besides classical logic, there are non-classical logics
  - minimal and intuitionistic logic ("constructive" logics)
  - modal logics, temporal logics like LTL (☞ Block 4), ...
  - many-valued logics, fuzzy logics
- Many of these logics come in different "levels"
  - Propositional logics (PL0)
  - First-order logics (PL1): (quantifiers over object variables)
  - Second-order (quantifiers over function/predicate symbols)
  - and even higher-order logics

Example: Optimization of if-then-else statements

original C code

```
if (!a && !b) h();
else if (!a) g();
else f();
```

```
if (!a) {
   if (!b) h();
   else g();
} else f();
```

optimized C code

```
if (a) f();
else if (b) g();
else h();
```

```
if (a) f();
else {
   if (!b) h();
   else g(); }
```

Example: Optimization of if-then-else statements

original C code

```
if (!a && !b) h();
else if (!a) g();
else f();
```

```
if (!a) {
   if (!b) h();
   else g();
} else f();
```

optimized C code

```
if (a) f();
else if (b) g();
else h();
```

```
if (a) f();
else {
   if (!b) h();
   else g(); }
```

Q: Is the optimized code equivalent to the original code?

Example: Optimization of if-then-else statements

original C code

```
if (!a && !b) h();
else if (!a) g();
else f();
```

↓

```
if (!a) {
  if (!b) h();
  else g();
} else f();
```

→

optimized C code

```
if (a) f();
else if (b) g();
else h();
```

↑

```
if (a) f();
else {
  if (!b) h();
  else g(); }
```

Q: Is the optimized code equivalent to the original code?

Translation of if-then-else statements

- Represent procedures as independent Boolean variables

original	optimized
if $\neg a \wedge \neg b$ *then* h	*if* a *then* f
else if $\neg a$ *then* g	*else if* b *then* g
else f	*else* h

- Recursively compile if-then-else into a propositional formula

$$compile(\textit{if } x \textit{ then } y \textit{ else } z) \equiv (x \wedge y) \vee (\neg x \wedge z)$$

- Check equivalence between original and optimized version

$$compile(\text{original}) \leftrightarrow compile(\text{optimized})$$

How to check equivalence?

- Compilation (+ simple manipulations) result in:

  $\neg a \wedge \neg b \wedge h \vee (a \vee b) \wedge (\neg a \wedge g \vee a \wedge f) \leftrightarrow a \wedge f \vee \neg a \wedge (b \wedge g \vee \neg b \wedge h)$

- Reformulate the problem as a SAT problem:

  Is there an assignment to a, b, f, g, h, which results in different evaluations of original and optimized?

- Or equivalently:

  Is the propositional formula

  $$\mathit{compile}(\text{original}) \quad \not\leftrightarrow \quad \mathit{compile}(\text{optimized})$$

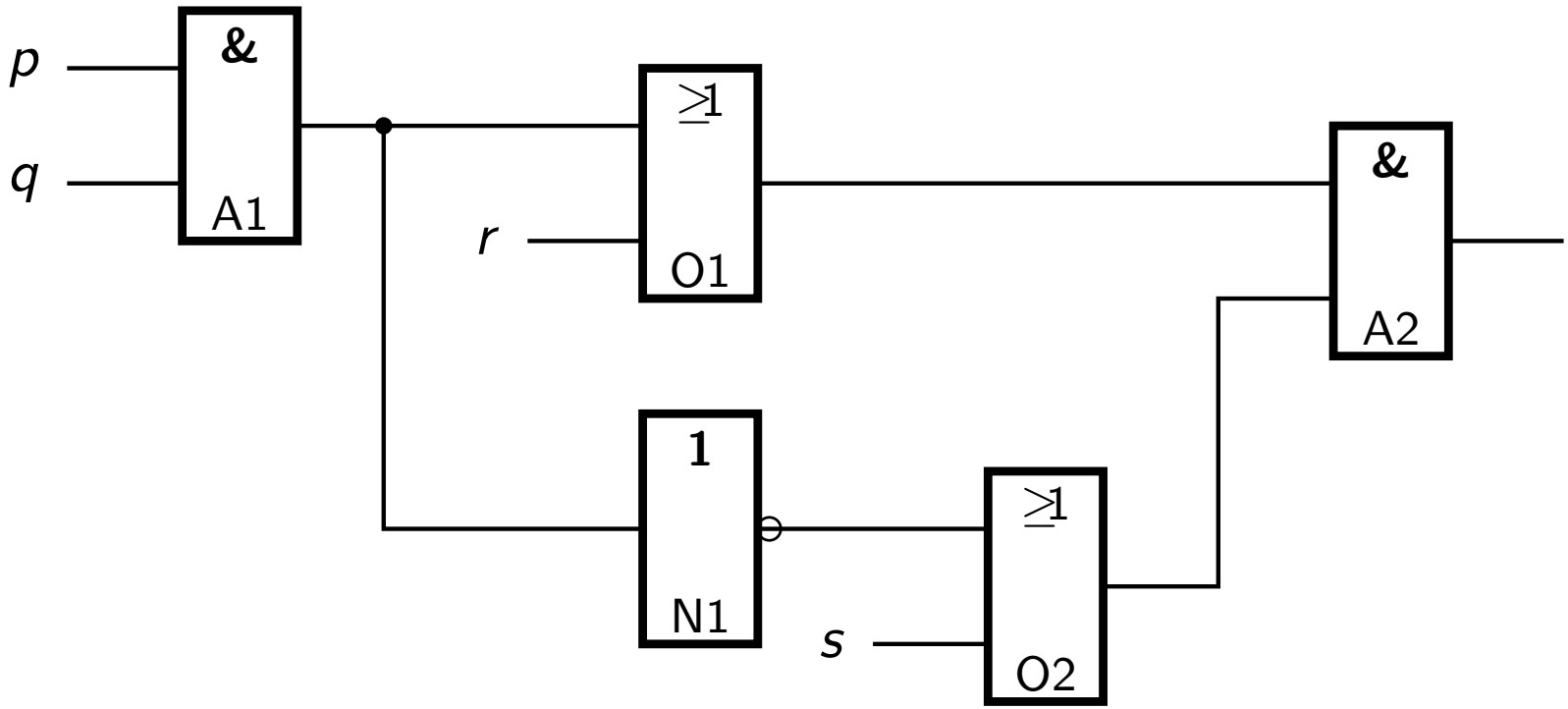
  satisfiable?

- A model provides a comprehensible counterexample

Example: Is the implementation of a circuit correct?

Does the implemented circuit compute the specified function?

$Spec$: $((p \wedge q) \vee r) \wedge (\neg(p \wedge q) \vee s)$

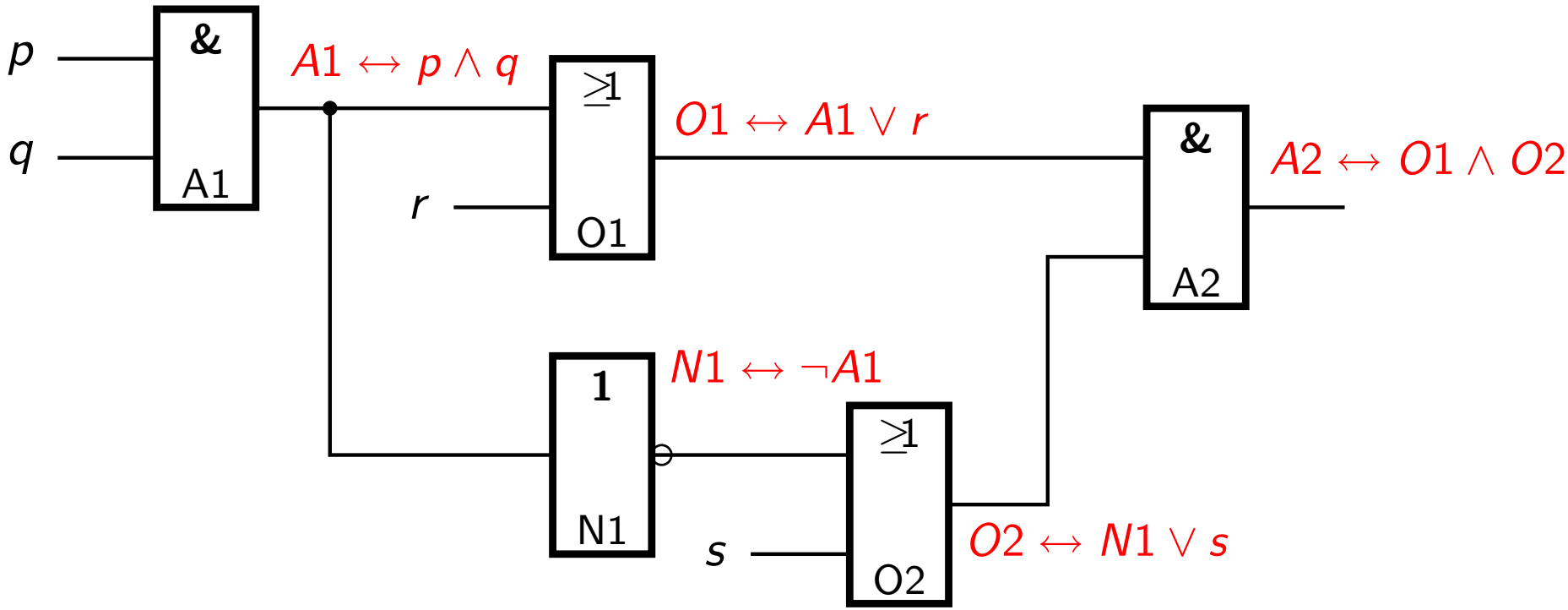


The definition of the gate symbols can be found at:
[Definition of gates (English)] [Definition of gates (German)]

Example: Is the implementation of a circuit correct?

Does the implemented circuit compute the specified function?

$Spec$: $((p \wedge q) \vee r) \wedge (\neg(p \wedge q) \vee s)$



Name each gate's output with a new variable and set the variable equivalent to the gate's input(s) and the gate's function!

How to model and solve the problem?

- Modeling: Abstract a given problem P into logic PL0 ($\leadsto P'$)
- Is the implementation correct (does *Impl* comply w. *Spec*)?
  - Take the circuit for *Impl* with output $A2$
  - Take *Spec* as a circuit with output B
  - Connect $A2$ and B to the 2 inputs of an XOR with output C
  - Check whether $C = 0$ for all input values of p, q, r, s
- P' is now a reasoning problem, not a test problem!
- Proof: Use automatic decision procedures to determine the solution of P, i.e., compute satisfiability of P'
  - SAT solvers and BDDs
  - Model checkers
  - Theorem provers

Outline

General information on Block 2

Introduction
- Motivation
- Examples
  - Equivalence checking of if-then-else statements
  - The circuit example

Syntax of propositional logic

Semantics of propositional logic
- Notations
- Entailment

Different normal forms and translation procedures

Learning Objectives

The syntax of propositional logic (alternative 1)

- Given a countable set, $\mathcal{BV}$, of Boolean variables
- Boolean variables like $p, q, r, p_1, \ldots$ represent facts:
  e.g., r represents "it is raining"
- Use $p, q, \ldots$ as metavariables for $\mathcal{BV}$s and $\varphi, \psi, \ldots$ as metavariables for formulas
- **Inductive definition of the set $\mathcal{L}$ of propositional formulas**

  B1: Every $p \in \mathcal{BV}$ is a formula (called atomic formula or atom)

  B2: $\top$ (verum) and $\bot$ (falsum) are formulas

  S1: If φ is a formula, then so is $(\neg\varphi)$ (negation)

  S2: If φ_1, φ_2 are formulas, then so are $(\varphi_1 \wedge \varphi_2)$ (conjunction), $(\varphi_1 \vee \varphi_2)$ (disjunction), $(\varphi_1 \rightarrow \varphi_2)$ (implication), $(\varphi_1 \leftrightarrow \varphi_2)$ (equivalence) and $(\varphi_1 \oplus \varphi_2)$ (exclusive-or)

NB: Bi is the ith base case, Si is the ith step case

The syntax of propositional logic (alternative 2)

$$\mathcal{L} ::= \top \mid \bot \mid \mathcal{BV} \mid (\neg\mathcal{L}) \mid (\mathcal{L} \wedge \mathcal{L}) \mid (\mathcal{L} \vee \mathcal{L}) \mid (\mathcal{L} \rightarrow \mathcal{L}) \mid (\mathcal{L} \leftrightarrow \mathcal{L}) \mid (\mathcal{L} \oplus \mathcal{L})$$

- Read this as: $\mathcal{L}$ is the smallest set such that 1. and 2. hold:
  1. If $p \in \mathcal{BV}$, p is $\top$ or p is $\bot$, then $p \in \mathcal{L}$
  2. If $\varphi, \psi \in \mathcal{L}$ then

     $(\neg\varphi)$, $(\varphi \wedge \psi)$, $(\varphi \vee \psi)$, $(\varphi \rightarrow \psi)$, $(\varphi \leftrightarrow \psi)$, $(\varphi \oplus \psi) \in \mathcal{L}$
- To save parenthesis, use the following ranking of binding strength: $\neg$, $\wedge$, $\vee$, $\rightarrow$, $\leftrightarrow$, $\oplus$ ($\neg$ binds stronger than $\wedge$, etc.)
- Example: $\neg p \wedge q \rightarrow r \vee s$ means $(((\neg p) \wedge q) \rightarrow (r \vee s))$

The syntax of propositional logic cont'd

Definition

1. A literal is $\top$, $\bot$, an atom or the negation thereof
2. A clause is a disjunction of literals
3. Immediate subformula (relation)
   - φ is an immediate subformula (isf) of $\neg\varphi$
   - φ_1 and φ_2 are isfs of $(\varphi_1 \circ \varphi_2)$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$
4. Subformula relation: reflexive-transitive closure of the immediate subformula relation

Example

Compute all subformulas of $\neg p \wedge q \rightarrow r \vee s$:

$\neg p \wedge q \rightarrow r \vee s$,

The syntax of propositional logic cont'd

Definition

1. A literal is $\top$, $\bot$, an atom or the negation thereof
2. A clause is a disjunction of literals
3. Immediate subformula (relation)
   - φ is an immediate subformula (isf) of $\neg\varphi$
   - φ_1 and φ_2 are isfs of $(\varphi_1 \circ \varphi_2)$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$
4. Subformula relation: reflexive-transitive closure of the immediate subformula relation

Example

Compute all subformulas of $\neg p \wedge q \rightarrow r \vee s$:

$\neg p \wedge q \rightarrow r \vee s$, $\neg p \wedge q$, $r \vee s$,

The syntax of propositional logic cont'd

Definition

1. A literal is $\top$, $\bot$, an atom or the negation thereof
2. A clause is a disjunction of literals
3. Immediate subformula (relation)
   - φ is an immediate subformula (isf) of $\neg\varphi$
   - φ_1 and φ_2 are isfs of $(\varphi_1 \circ \varphi_2)$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$
4. Subformula relation: reflexive-transitive closure of the immediate subformula relation

Example

Compute all subformulas of $\neg p \wedge q \rightarrow r \vee s$:

$\neg p \wedge q \rightarrow r \vee s$, $\neg p \wedge q$, $r \vee s$, $\neg p$, q,

The syntax of propositional logic cont'd

Definition

1. A literal is $\top$, $\bot$, an atom or the negation thereof
2. A clause is a disjunction of literals
3. Immediate subformula (relation)
   - φ is an immediate subformula (isf) of $\neg\varphi$
   - φ_1 and φ_2 are isfs of $(\varphi_1 \circ \varphi_2)$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$
4. Subformula relation: reflexive-transitive closure of the immediate subformula relation

Example

Compute all subformulas of $\neg p \wedge q \rightarrow r \vee s$:

$\neg p \wedge q \rightarrow r \vee s$, $\neg p \wedge q$, $r \vee s$, $\neg p$, q, r, s,

The syntax of propositional logic cont'd

Definition

1. A literal is $\top$, $\bot$, an atom or the negation thereof
2. A clause is a disjunction of literals
3. Immediate subformula (relation)
   - φ is an immediate subformula (isf) of $\neg\varphi$
   - φ_1 and φ_2 are isfs of $(\varphi_1 \circ \varphi_2)$ for $\circ \in \{\wedge, \vee, \rightarrow, \leftrightarrow, \oplus\}$
4. Subformula relation: reflexive-transitive closure of the immediate subformula relation

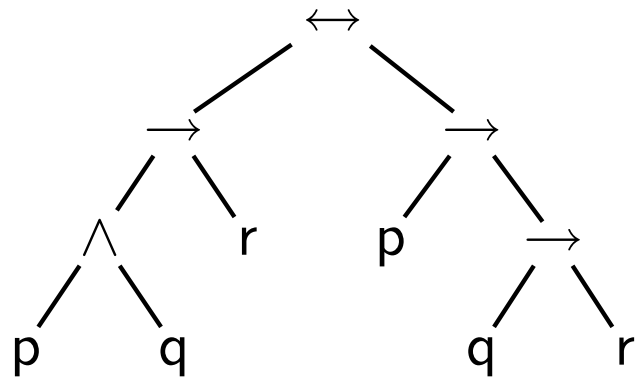
Example

Compute all subformulas of $\neg p \wedge q \rightarrow r \vee s$:

$\neg p \wedge q \rightarrow r \vee s$, $\neg p \wedge q$, $r \vee s$, $\neg p$, q, r, s, p

Propositional formulas as trees

- Formulas can be depicted as formula trees
- Example: $((p \wedge q) \rightarrow r) \leftrightarrow (p \rightarrow (q \rightarrow r))$



Outline

General information on Block 2

Introduction
- Motivation
- Examples
  - Equivalence checking of if-then-else statements
  - The circuit example

Syntax of propositional logic

Semantics of propositional logic
- Notations
- Entailment

Different normal forms and translation procedures

Learning Objectives

The semantics of propositional logic

Definition

1. A Boolean value (or truth value) is either 0 (false) or 1 (true)
2. An interpretation function for a set $\mathcal{P}$ of Boolean variables:

$$\text{mapping} \quad I : \mathcal{P} \mapsto \{0, 1\}$$

Interpretation functions are often called truth assignments, e.g., I assigns 1 to $p \in \mathcal{P}$, i.e., $I(p) = 1$

Since we want to "evaluate" formulas under I, we have to extend I to formulas

The extension of I to formulas

- $I(\top) = 1$ and $I(\bot) = 0$
- $I(\neg\varphi) = 1$ iff $I(\varphi) = 0$
- $I(\varphi \wedge \psi) = 1$ iff $I(\varphi) = I(\psi) = 1$
- $I(\varphi \vee \psi) = 1$ iff $I(\varphi) = 1$ or $I(\psi) = 1$
- $I(\varphi \to \psi) = 1$ iff $I(\varphi) = 0$ or $I(\psi) = 1$
- $I(\varphi \leftrightarrow \psi) = 1$ iff $I(\varphi) = I(\psi)$
- $I(\varphi \oplus \psi) = 1$ iff $I(\varphi) \neq I(\psi)$

Equivalent notations:

φ is true under I	iff I satisfies φ	iff $I(\varphi) = 1$	iff $I \models \varphi$
φ is false under I	iff I does not satisfy φ	iff $I(\varphi) = 0$	iff $I \not\models \varphi$

Some notations

- If I satisfies φ, then we call I a model of φ
- $Mod(\psi)$ is the class of all models of ψ

- φ is satisfiable (valid) if φ is true in some (all) interpretations
- φ is unsatisfiable if φ is false in all interpretations
- Formulas φ and ψ are equivalent, denoted by $\varphi \equiv \psi$, iff they have exactly the same models, i.e., $Mod(\varphi) = Mod(\psi)$

- Example: $(\varphi \to \psi) \equiv (\neg\psi \to \neg\varphi)$: Take $I \in Mod(\varphi \to \psi)$

$$
\begin{aligned}
I(\varphi \to \psi) = 1 \quad & \text{iff } I(\varphi) = 0 \text{ or } I(\psi) = 1 \\
& \text{iff } I(\neg\varphi) = 1 \text{ or } I(\psi) = 1 \\
& \text{iff } I(\neg\varphi) = 1 \text{ or } I(\neg\psi) = 0 \\
& \text{iff } I(\neg\psi \to \neg\varphi) = 1
\end{aligned}
$$

Some notations

- If I satisfies φ, then we call I a model of φ
- $Mod(\psi)$ is the class of all models of ψ

- φ is satisfiable (valid) if φ is true in some (all) interpretations
- φ is unsatisfiable if φ is false in all interpretations
- Formulas φ and ψ are equivalent, denoted by $\varphi \equiv \psi$, iff they have exactly the same models, i.e., $Mod(\varphi) = Mod(\psi)$

- Example: $(\varphi \to \psi) \equiv (\neg\psi \to \neg\varphi)$: Take $I \in Mod(\varphi \to \psi)$

$$
\begin{aligned}
I(\varphi \to \psi) = 1 \quad & \text{iff } I(\varphi) = 0 \text{ or } I(\psi) = 1 \\
& \text{iff } I(\neg\varphi) = 1 \text{ or } I(\psi) = 1 \\
& \text{iff } I(\neg\varphi) = 1 \text{ or } I(\neg\psi) = 0 \\
& \text{iff } I(\neg\psi \to \neg\varphi) = 1
\end{aligned}
$$

Did we prove it for all $I \in Mod(\varphi \to \psi)$? Explain in detail!

Entailment (or logical implication)

- So far, $\models$ relates an interpretation and a formula
- Allow also a set of formulas on the left side
- Important: a set of formulas coincides with the conjunction of its elements, i.e., $\{\varphi_1, \ldots, \varphi_n\}$ is $\bigwedge_{i=1}^{n} \varphi_i$
- Important: an empty conjunction is 1 in all interpretations i.e., it is equivalent to $\top$
- W entails φ, $W \models \varphi$, iff $Mod(W) \subseteq Mod(\varphi)$
- $W \models \varphi$ iff $I \models \varphi$ for all models I of W (for all $I \in Mod(W)$)

Entailment becomes important when we consider theories

Reduction to satisfiability

Reduce validity, entailment, equivalence to satisfiability

1. Validity
   - $\neg\varphi$ is unsatisfiable iff φ is valid
2. Entailment
   - φ entails ψ ($\varphi \models \psi$) iff $\varphi \rightarrow \psi$ is valid (apply Deduction Thm)
   - Hence, $\varphi \models \psi$ iff $\varphi \wedge \neg\psi$ (i.e., $\neg(\varphi \rightarrow \psi)$) is unsatisfiable
3. Equivalence
   - φ is equivalent to ψ ($\varphi \equiv \psi$) iff $\varphi \leftrightarrow \psi$ is valid
   - Hence, $\varphi \equiv \psi$ iff $\varphi \models \psi$ and $\psi \models \varphi$ hold
   - Consequently, $\varphi \equiv \psi$ iff $\varphi \wedge \neg\psi$ and $\psi \wedge \neg\varphi$ are unsatisfiable

A sound and complete procedure for satisfiability is sufficient!

Recap: The SAT problem

Given a propositional formula φ over n propositional variables $V = \{x_1, \ldots, x_n\}$.
Is there an assignment $I\colon V \mapsto \{0, 1\}$ with $I(\varphi) = 1$?

- SAT belongs to NP, i.e., there is a non-deterministic Turing machine deciding SAT in PTIME
  - which guesses an assignment I (linear in n) and
  - calculates $I(\varphi)$ (linear in $|\varphi|$)
- SAT is complete for NP (see the complexity part of this lecture)
- General SAT algorithms are exponential in time in the worst case (unless P=NP), but they work well in practice!

Outline

General information on Block 2

Introduction
- Motivation
- Examples
  - Equivalence checking of if-then-else statements
  - The circuit example

Syntax of propositional logic

Semantics of propositional logic
- Notations
- Entailment

Different normal forms and translation procedures

Learning Objectives

Normal forms: NNF and CNF

Restricted syntax, but retain the important properties (application-dependent, e.g., provability or satisfiability)

Definition

Negation Normal Form (NNF): A formula in NNF has

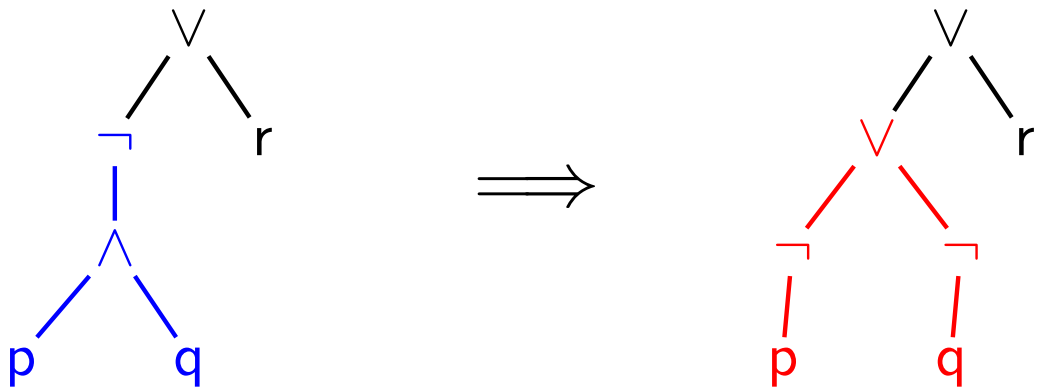
1. negation signs only in front of atoms and
2. the only connectives are $\wedge$ and $\vee$.

Conjunctive Normal Form (CNF): A formula in CNF consists of a conjuction of clauses (=disjunction of literals). A CNF is often represented as a set of clauses.

Fact: Every propositional formula has an equivalent formula in NNF/CNF! But how do we construct them?

Preparatory concept: Tree replacements

- Let $\varphi[\psi]$ denote that ψ occurs in φ
- This means that ψ occurs as zero, one or more subtree(s) in the formula tree of φ
- Possibility to perform subtree replacements
- Example: $\varphi\colon \neg(p \wedge q) \vee r$ and $\psi_1\colon \neg(p \wedge q)$
  - Then $\varphi[\psi_1]$ indicates occurrence(s) of ψ_1 in φ
  - Construct $\varphi[\psi_2]$ with $\psi_2\colon \neg p \vee \neg q$ by a tree replacement



Equivalent replacement

Q: Under which condition(s) does $\varphi[\psi_1] \equiv \varphi[\psi_2]$ hold?

Lemma (Equivalent Replacement Lemma)

Let I be an assignment and $I \models \psi_1 \leftrightarrow \psi_2$. Then $I \models \varphi[\psi_1] \leftrightarrow \varphi[\psi_2]$.

Theorem (Equivalent Replacement Theorem (ERT))

Let $\psi_1 \equiv \psi_2$. Then $\varphi[\psi_1] \equiv \varphi[\psi_2]$.

Basics of the NNF translation

Definition (Translation of a propositional formula to NNF)

Perform the following steps:

1. Replace all $\leftrightarrow$ by $\rightarrow$ using $(\varphi \leftrightarrow \psi) \equiv ((\varphi \rightarrow \psi) \wedge (\psi \rightarrow \varphi))$
2. Replace all $\oplus$ using $(\varphi \oplus \psi) \equiv ((\varphi \vee \psi) \wedge (\neg\varphi \vee \neg\psi))$ (why?)
3. Replace all $\rightarrow$ using $(\varphi \rightarrow \psi) \equiv (\neg\varphi \vee \psi)$
4. Replace the left side of the following equivalences by the right side (order does not matter!)

$$\begin{array}{rcl} \neg(\varphi \vee \psi) & \equiv & \neg\varphi \wedge \neg\psi \\ \varphi \vee \top & \equiv & \top \\ \varphi \wedge \bot & \equiv & \bot \\ \varphi \wedge \top & \equiv & \varphi \\ \varphi \vee \bot & \equiv & \varphi \\ \neg\neg\varphi & \equiv & \varphi \end{array} \qquad \begin{array}{rcl} \neg(\varphi \wedge \psi) & \equiv & \neg\varphi \vee \neg\psi \\ \top \vee \varphi & \equiv & \top \\ \bot \wedge \varphi & \equiv & \bot \\ \top \wedge \varphi & \equiv & \varphi \\ \bot \vee \varphi & \equiv & \varphi \end{array}$$

Basics of the NNF translation

Fact: The translation process terminates and produces the NNF. A proof can be found in A. Leitsch: The resolution calculus. Springer, 1997. [link to Springer]

By employing the Equivalent Replacement Theorem, we obtain:

For any propositional formula λ, $nnf(\lambda) \equiv \lambda$ holds.

Basics of the CNF translation

- Based on the application of distributivity laws
- Start with the formula λ and translate it to NNF
- Take $\mathit{nnf}(\lambda)$ and replace the left side of the following equivalences by the right side (order does not matter!)

1	$\varphi \vee (\psi \wedge \chi)$	$\equiv$	$(\varphi \vee \psi) \wedge (\varphi \vee \chi)$
2	$(\psi \wedge \chi) \vee \varphi$	$\equiv$	$(\psi \vee \varphi) \wedge (\chi \vee \varphi)$

- Observe that $\mathit{nnf}(\lambda) \equiv \mathit{cnf}(\lambda) \equiv \lambda$ holds Why?

Exa: Transform φ: $(p \wedge q \rightarrow r) \rightarrow (q \rightarrow r)$ to CNF

$nnf(\varphi)$: $(p \wedge q \wedge \neg r) \vee (\neg q \vee r)$ (why?)

Formula	Rule
$(p \wedge q \wedge \neg r) \vee (\neg q \vee r)$	2
$(p \vee \neg q \vee r) \wedge (q \wedge \neg r \vee (\neg q \vee r))$	2
$(p \vee \neg q \vee r) \wedge (q \vee \neg q \vee r) \wedge (\neg r \vee \neg q \vee r)$	

This transformation has two disadvantages:

1. Disruption of the formula's structure
2. $cnf(\varphi)$ can be exponentially longer than φ
   Take a formula in disjunctive NF and translate it to CNF!

Can be avoided: use definitional translation ($\rightsquigarrow$ circuit exa)

Structure-preserving (or definitional) NFTs

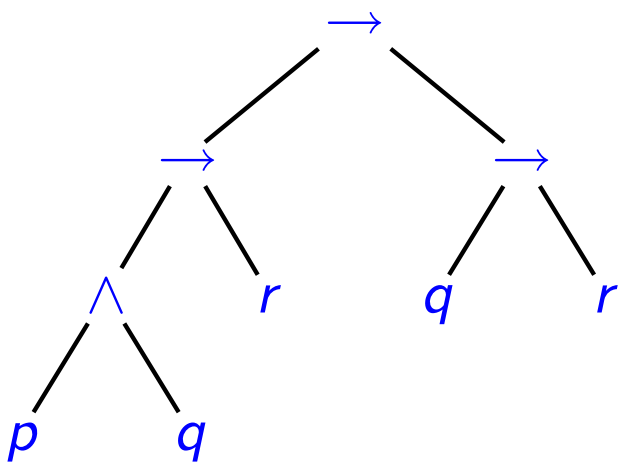
The basic idea

- Well known in logic
  (occurred relatively late in ATP and theory (Tseitin 1968))
- Consider the input formula φ as a tree or a dag (as before)
- Label each subformula occurrence (SFO) with a new atom
  (atom neither occurs in φ nor is it introduced before)
- Construct equivalences of the form
  $$\ell_\varphi \leftrightarrow (\ell_{\varphi_1} \circ \ell_{\varphi_2}) \quad \text{for SFOs} \quad \varphi_1 \circ \varphi_2$$
  where ℓ_ψ is the label for SF(O) ψ.
- Translate each $\ell_\varphi \leftrightarrow (\ell_{\varphi_1} \circ \ell_{\varphi_2})$ to CNF
- Such an NFT is called a Tseitin translation

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

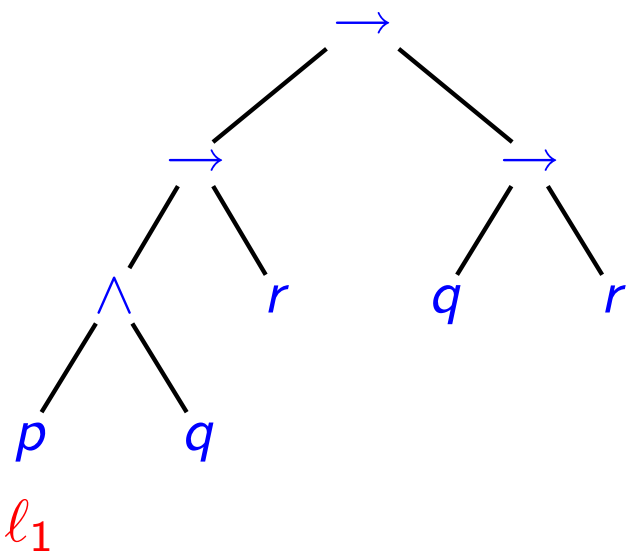
Tree of φ



Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

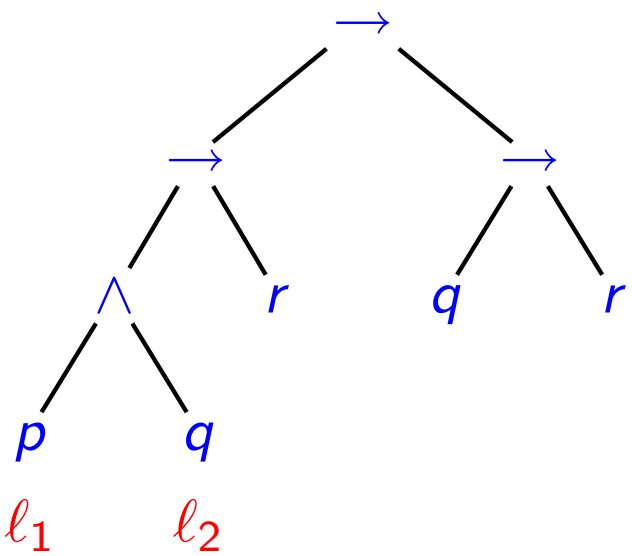


$D_1\colon \ell_1 \quad \leftrightarrow \quad p$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ



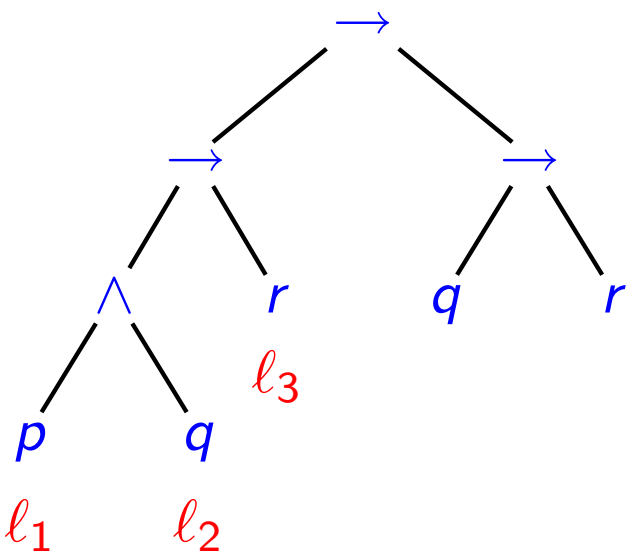
$D_1 : \ell_1 \quad \leftrightarrow \quad p$

$D_2 : \ell_2 \quad \leftrightarrow \quad q$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

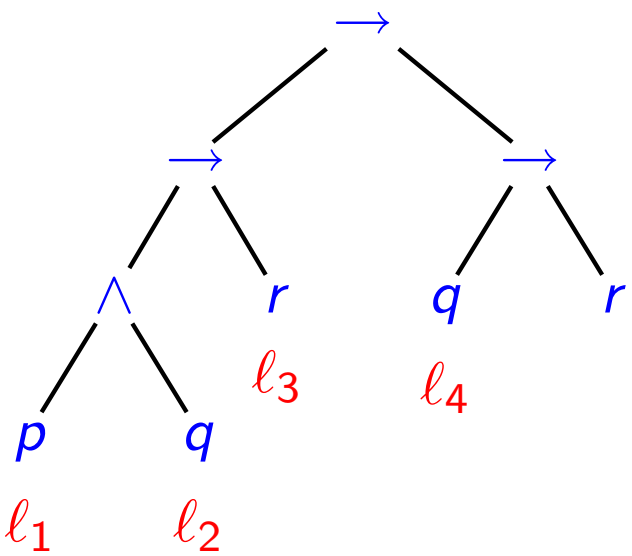


$$D_1\colon \ell_1 \quad \leftrightarrow \quad p$$
$$D_2\colon \ell_2 \quad \leftrightarrow \quad q$$
$$D_3\colon \ell_3 \quad \leftrightarrow \quad r$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

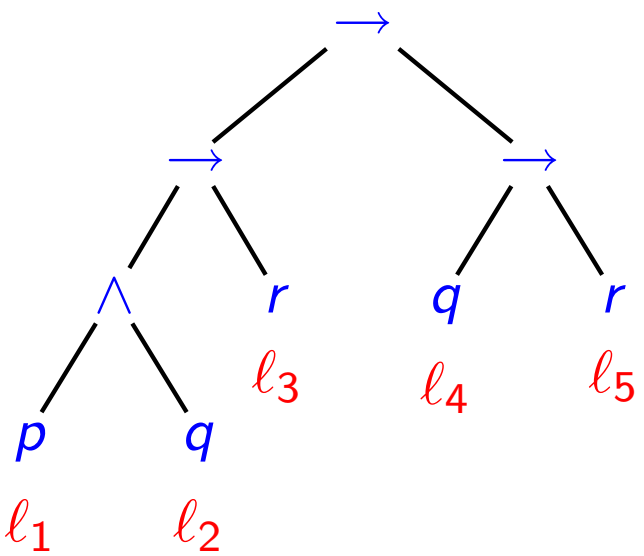


$$D_1 : \ell_1 \leftrightarrow p$$
$$D_2 : \ell_2 \leftrightarrow q$$
$$D_3 : \ell_3 \leftrightarrow r$$
$$D_4 : \ell_4 \leftrightarrow q$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ



$D_1 : \ell_1 \quad \leftrightarrow \quad p$

$D_2 : \ell_2 \quad \leftrightarrow \quad q$

$D_3 : \ell_3 \quad \leftrightarrow \quad r$

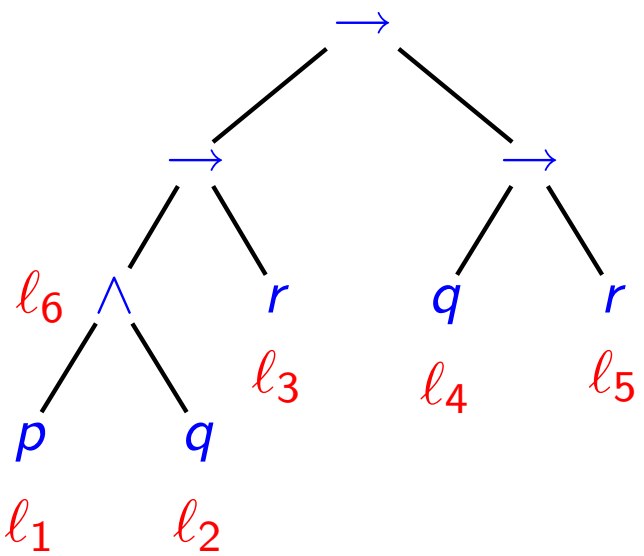
$D_4 : \ell_4 \quad \leftrightarrow \quad q$

$D_5 : \ell_5 \quad \leftrightarrow \quad r$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

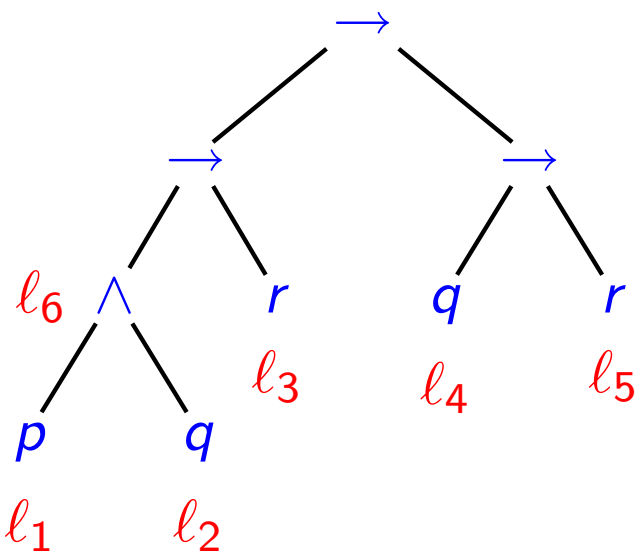


$$D_1 : \ell_1 \leftrightarrow p$$
$$D_2 : \ell_2 \leftrightarrow q$$
$$D_3 : \ell_3 \leftrightarrow r$$
$$D_4 : \ell_4 \leftrightarrow q$$
$$D_5 : \ell_5 \leftrightarrow r$$
$$D_6 : \ell_6 \leftrightarrow p \wedge q$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

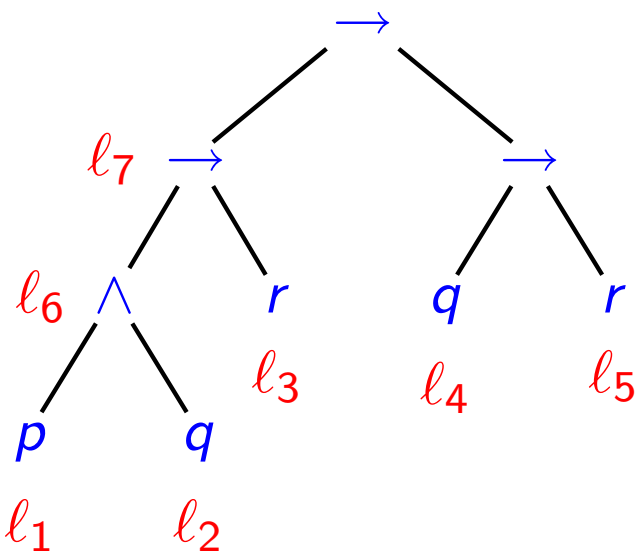


$$D_1 : \ell_1 \leftrightarrow p$$
$$D_2 : \ell_2 \leftrightarrow q$$
$$D_3 : \ell_3 \leftrightarrow r$$
$$D_4 : \ell_4 \leftrightarrow q$$
$$D_5 : \ell_5 \leftrightarrow r$$
$$D_6 : \ell_6 \leftrightarrow \ell_1 \wedge \ell_2$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

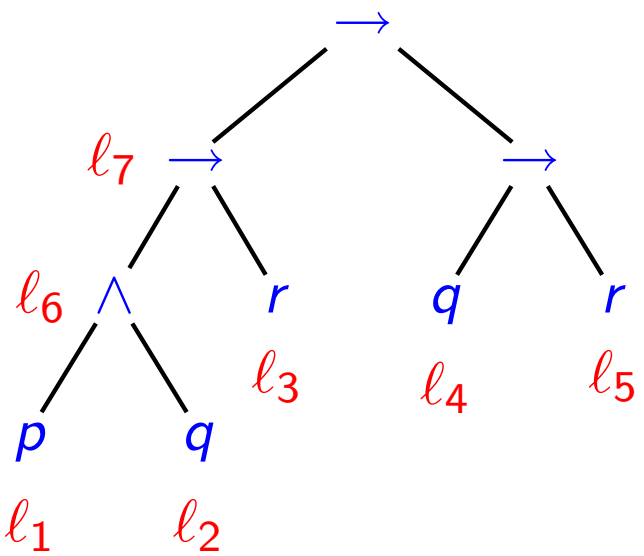


$$D_1 : \ell_1 \quad \leftrightarrow \quad p$$
$$D_2 : \ell_2 \quad \leftrightarrow \quad q$$
$$D_3 : \ell_3 \quad \leftrightarrow \quad r$$
$$D_4 : \ell_4 \quad \leftrightarrow \quad q$$
$$D_5 : \ell_5 \quad \leftrightarrow \quad r$$
$$D_6 : \ell_6 \quad \leftrightarrow \quad \ell_1 \wedge \ell_2$$
$$D_7 : \ell_7 \quad \leftrightarrow \quad p \wedge q \longrightarrow r$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

Tree of φ

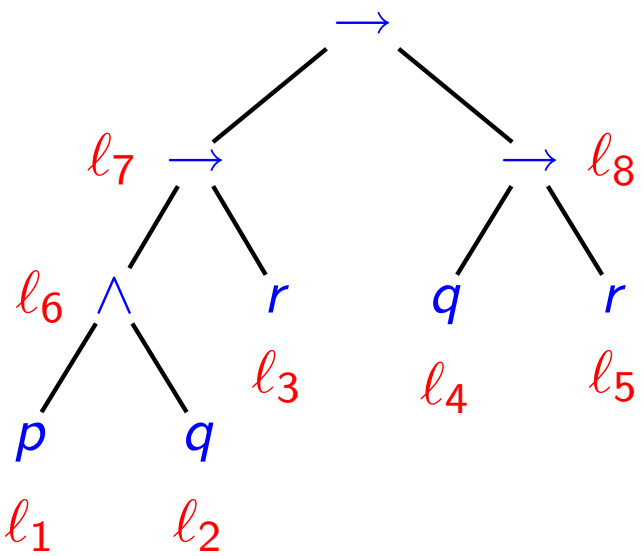


$$D_1 : \ell_1 \leftrightarrow p$$
$$D_2 : \ell_2 \leftrightarrow q$$
$$D_3 : \ell_3 \leftrightarrow r$$
$$D_4 : \ell_4 \leftrightarrow q$$
$$D_5 : \ell_5 \leftrightarrow r$$
$$D_6 : \ell_6 \leftrightarrow \ell_1 \wedge \ell_2$$
$$D_7 : \ell_7 \leftrightarrow \ell_6 \longrightarrow \ell_3$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs

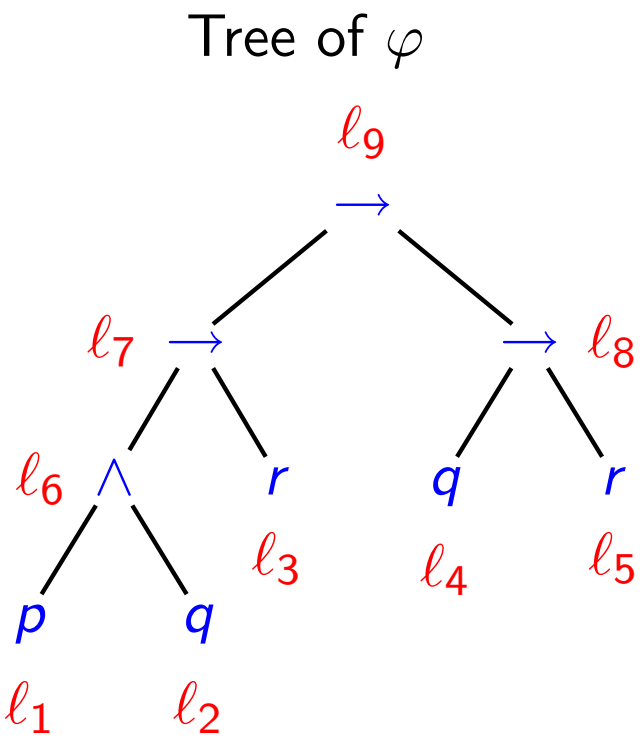
Tree of φ



$$
\begin{aligned}
D_1 &: \ell_1 \leftrightarrow p \\
D_2 &: \ell_2 \leftrightarrow q \\
D_3 &: \ell_3 \leftrightarrow r \\
D_4 &: \ell_4 \leftrightarrow q \\
D_5 &: \ell_5 \leftrightarrow r \\
D_6 &: \ell_6 \leftrightarrow \ell_1 \wedge \ell_2 \\
D_7 &: \ell_7 \leftrightarrow \ell_6 \longrightarrow \ell_3 \\
D_8 &: \ell_8 \leftrightarrow \ell_4 \longrightarrow \ell_5
\end{aligned}
$$

Example for a translation: $\varphi\colon (p \wedge q \longrightarrow r) \longrightarrow (q \longrightarrow r)$

Step 1: Annotate the formula tree and generate equivalences for SFOs



$$
\begin{aligned}
D_1 &: \ell_1 \leftrightarrow p \\
D_2 &: \ell_2 \leftrightarrow q \\
D_3 &: \ell_3 \leftrightarrow r \\
D_4 &: \ell_4 \leftrightarrow q \\
D_5 &: \ell_5 \leftrightarrow r \\
D_6 &: \ell_6 \leftrightarrow \ell_1 \wedge \ell_2 \\
D_7 &: \ell_7 \leftrightarrow \ell_6 \longrightarrow \ell_3 \\
D_8 &: \ell_8 \leftrightarrow \ell_4 \longrightarrow \ell_5 \\
D_9 &: \ell_9 \leftrightarrow \ell_7 \longrightarrow \ell_8
\end{aligned}
$$

The original formula and results of the translation

$$\varphi \text{ is valid} \quad \text{iff} \quad \left(\bigwedge_{i=1}^{9} D_i \right) \rightarrow \ell_9 \text{ is valid} \quad \text{iff} \quad \bigwedge_{i=1}^{9} D_i \wedge \neg \ell_9 \text{ is unsatisfiable}$$

$$\varphi \text{ is satisfiable} \quad \text{iff} \quad \bigwedge_{i=1}^{9} D_i \wedge \ell_9 \text{ is satisfiable}$$

Some proofs will be discussed in the exercise part.

Example for a translation: $\varphi\colon (p \wedge q \rightarrow r) \rightarrow (q \rightarrow r)$

Step 2: Translate the "labeling formulas" to clauses

Equivalences for SFOs in φ	Associated Clauses $C_1(\varphi)$	$C_2(\varphi)$	$C_3(\varphi)$
$\ell_1 \leftrightarrow p$	$\neg\ell_1 \vee p$	$\ell_1 \vee \neg p$	$\top$
$\ell_2 \leftrightarrow q$	$\neg\ell_2 \vee q$	$\ell_2 \vee \neg q$	$\top$
$\ell_3 \leftrightarrow r$	$\neg\ell_3 \vee r$	$\ell_3 \vee \neg r$	$\top$
$\ell_4 \leftrightarrow q$	$\neg\ell_4 \vee q$	$\ell_4 \vee \neg q$	$\top$
$\ell_5 \leftrightarrow r$	$\neg\ell_5 \vee r$	$\ell_5 \vee \neg r$	$\top$
$\ell_6 \leftrightarrow \ell_1 \wedge \ell_2$	$\neg\ell_6 \vee \ell_1$	$\neg\ell_6 \vee \ell_2$	$\ell_6 \vee \neg\ell_1 \vee \neg\ell_2$
$\ell_7 \leftrightarrow \ell_6 \rightarrow \ell_3$	$\neg\ell_7 \vee \neg\ell_6 \vee \ell_3$	$\ell_7 \vee \ell_6$	$\ell_7 \vee \neg\ell_3$
$\ell_8 \leftrightarrow \ell_4 \rightarrow \ell_5$	$\neg\ell_8 \vee \neg\ell_4 \vee \ell_5$	$\ell_8 \vee \ell_4$	$\ell_8 \vee \neg\ell_5$
$\ell_9 \leftrightarrow \ell_7 \rightarrow \ell_8$	$\neg\ell_9 \vee \neg\ell_7 \vee \ell_8$	$\ell_9 \vee \ell_7$	$\ell_9 \vee \neg\ell_8$

The structure-preserving normal forms

The definitional form, $\delta(\varphi)$, of φ is $\hat{\delta}(\varphi) \cup \{\ell_\varphi\}$ with

$$\hat{\delta}(\varphi) \quad : \quad \{C_1(\psi), C_2(\psi) \mid \psi \in \Sigma(\varphi)\} \cup \{C_3(\psi) \mid \psi \in \Sigma(\varphi) \wedge C_3(\psi) \neq \top\}$$

Tseitin

- $\Sigma(\varphi)$: Set of all SFOs of φ
- Sat-equivalence: φ has a model iff $\delta(\varphi)$ has a model
- Several variants and optimizations available, e.g.,
  - no label for atoms or negations,
  - don't translate clauses,
  - only one implication instead of an equivalence depending on the polarity of the subformula occurrence, etc.

    ☞ results in the translation of Plaisted and Greenbaum

Properties of the definitional translation

- Retains the structure of the formula (by labels for SFOs)
- For each SFOs, there are at most four clauses
- Each clause has at most three literals
- Normal form is linear-time computable
- φ and its definitional translation not logically equivalent
  (new labels change signature $\rightsquigarrow$ logical equivalence lost)
- φ has a model iff its definitional translation has one holds
  (this is the important prop.; cf 1st order ATP and Skolemization)
- Models of φ and its def. translation can be translated
- Generalizations used to extend calculi by extensions
  (resulting in stronger calculi which p-simulate, e.g., the cut rule)

The DIMACS CNF input format

- Standard input format for most SAT solver
- Plain text file of the form:

```
p cnf <no_of_variables> <no_of_clauses>
<clause> 0
<clause> 0
...
```

- One or more lines per clause

The DIMACS CNF input format: Clauses

- A clause is a list of non-zero numbers separated by spaces
- Each clause is terminated by 0
- A positive number correspond to a variable
  (propositional variables have to be mapped to positive integers)
- A negative number correspond to a negated variable

If you would like to try a SAT solver, consider

- minisat [link] or
- lingeling and plingeling [link]

Learning objectives

- Ability to discuss the overall procedure and motivate the use of SAT
- Ability to discuss connections to the other parts of the lecture
- Ability to model simple problems in PL0
- Ability to reduce reasoning problems to others
- Ability to convert PL0 formulas/circuits to different NFs
- Ability to generate the DIMACS format from a CNF
- Ability to prove basic properties about NFs (ex part)
- Ability to explain in detail and perform proofs using models (e.g., equivalences or entailments)