IN3070/4070 – Logic – Autumn 2020

Lecture 1: Introduction

Martin Giese

20th August 2020

Department of Informatics

University of Oslo

Today's Plan

- What is Logic?
- Logic in Computer Science
- Three Ingredients
- Applications
- Course Information

Outline

- What is Logic?
- Logic in Computer Science
- Three Ingredients
- Applications
- Course Information

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o\gamma\iota\kappa\acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o \gamma \iota \kappa \acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o \gamma \iota \kappa \acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

Richard Smullyan

"To make precise the notion of a proof."

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o \gamma \iota \kappa \acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

Richard Smullyan

"To make precise the notion of a proof."

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o\gamma\iota\kappa\acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

Richard Smullyan

"To make precise the notion of a proof."

Bertrand Russell

"The subject in which nobody knows what one is talking about, nor whether what one is saying is true."

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o \gamma \iota \kappa \acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

Richard Smullyan

"To make precise the notion of a proof."

Bertrand Russell

"The subject in which nobody knows what one is talking about, nor whether what one is saying is true."

Confusing... let's be computer scientists and compute something!

What is Logic?

Wikipedia

Logic (from the Ancient Greek: $\lambda o \gamma \iota \kappa \acute{\eta}$) is the systematic study of the form of valid inference, and the most general laws of truth.

Richard Smullyan

"To make precise the notion of a proof."

Bertrand Russell

"The subject in which nobody knows what one is talking about, nor whether what one is saying is true."

Confusing... let's be computer scientists and compute something!

Computation

- What is computation?

Computation

- What is computation?

$$\frac{\begin{array}{c}A \text{ owns } x\ B\text{s} \\ A \text{ gets another } y\ B\text{s}\end{array}}{A \text{ now owns } (x+y)\ B\text{s}}$$

Computation

- What is computation?

$$\frac{A \text{ owns } x\ B\text{s} \qquad A \text{ gets another } y\ B\text{s}}{A \text{ now owns } (x+y)\ B\text{s}}$$

e.g.

$$\frac{\text{Peter owns 1 apple} \qquad \text{Peter gets another 4 apples}}{\text{Peter now owns 5 apples}}$$

Computation

- What is computation?

$$\frac{\begin{array}{c} A \text{ owns } x\ B\text{s} \\ A \text{ gets another } y\ B\text{s} \end{array}}{A \text{ now owns } (x+y)\ B\text{s}}$$

e.g.

$$\frac{\begin{array}{c} \text{Peter owns 1 apple} \\ \text{Peter gets another 4 apples} \end{array}}{\text{Peter now owns 5 apples}}$$

- Computation is algorithmic manipulation of numbers...

Computation

- What is computation?

$$\frac{A \text{ owns } x \ B\text{s} \qquad A \text{ gets another } y \ B\text{s}}{A \text{ now owns } (x+y) \ B\text{s}}$$

e.g.

$$\frac{\text{Peter owns 1 apple} \qquad \text{Peter gets another 4 apples}}{\text{Peter now owns 5 apples}}$$

- Computation is algorithmic manipulation of numbers...
- ...where the *meaning* of the numbers is not needed

Computation

- What is computation?

$$\frac{A \text{ owns } x\ B\text{s} \qquad A \text{ gets another } y\ B\text{s}}{A \text{ now owns } (x+y)\ B\text{s}}$$

e.g.

$$\frac{\text{Peter owns 1 apple} \qquad \text{Peter gets another 4 apples}}{\text{Peter now owns 5 apples}}$$

- Computation is algorithmic manipulation of numbers...
- ...where the *meaning* of the numbers is not needed
- Can compute $1 + 4 = 5$ without knowing what is counted

Computation

- What is computation?

$$\frac{\begin{array}{c} A \text{ owns } x\ B\text{s} \\ A \text{ gets another } y\ B\text{s} \end{array}}{A \text{ now owns } (x+y)\ B\text{s}}$$

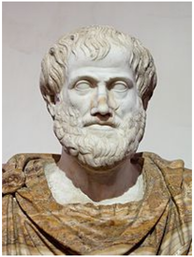
e.g.

$$\frac{\begin{array}{c} \text{Peter owns 1 apple} \\ \text{Peter gets another 4 apples} \end{array}}{\text{Peter now owns 5 apples}}$$

- Computation is algorithmic manipulation of numbers. . .
- . . . where the *meaning* of the numbers is not needed
- Can compute $1 + 4 = 5$ without knowing what is counted
- Abstraction!

Computation with Knowledge

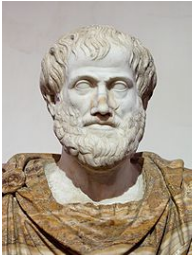
- Can be traced back to Aristotle (384–322 BC)



Computation with Knowledge

- Can be traced back to Aristotle (384–322 BC)
- Modus Barbara:

$$\frac{\begin{array}{c}\text{All } A \text{ are } B\\ \text{All } B \text{ are } C\end{array}}{\text{All } A \text{ are } C}$$



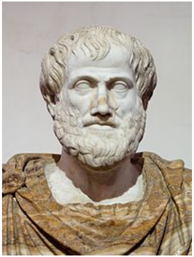
Computation with Knowledge

- Can be traced back to Aristotle (384–322 BC)
- Modus Barbara:

$$\frac{\begin{array}{c}\text{All } A \text{ are } B\\ \text{All } B \text{ are } C\end{array}}{\text{All } A \text{ are } C}$$

e.g.

$$\frac{\begin{array}{c}\text{All Greeks are men}\\ \text{All men are mortal}\end{array}}{\text{All Greeks are mortal}}$$



Computation with Knowledge

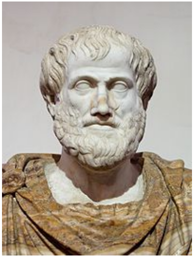
- Can be traced back to Aristotle (384–322 BC)
- Modus Barbara:

$$\frac{\begin{array}{c}\text{All } A \text{ are } B\\ \text{All } B \text{ are } C\end{array}}{\text{All } A \text{ are } C}$$

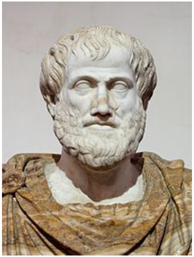
e.g.

$$\frac{\begin{array}{c}\text{All Greeks are men}\\ \text{All men are mortal}\end{array}}{\text{All Greeks are mortal}}$$

- Algorithmic manipulation of *knowledge*...



Computation with Knowledge



- Can be traced back to Aristotle (384–322 BC)
- Modus Barbara:

$$\frac{\text{All } A \text{ are } B \qquad \text{All } B \text{ are } C}{\text{All } A \text{ are } C}$$

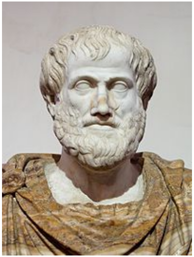
All *A* are *B*
All *B* are *C*
———
All *A* are *C*

e.g.

All Greeks are men
All men are mortal
———
All Greeks are mortal

- Algorithmic manipulation of *knowledge*. . .
- . . . where the *meaning* of the words is not needed!

Computation with Knowledge



- Can be traced back to Aristotle (384–322 BC)
- Modus Barbara:

$$\frac{\text{All } A \text{ are } B \qquad \text{All } B \text{ are } C}{\text{All } A \text{ are } C}$$

e.g.

$$\frac{\text{All Greeks are men} \qquad \text{All men are mortal}}{\text{All Greeks are mortal}}$$

- Algorithmic manipulation of *knowledge*. . .
- . . . where the *meaning* of the words is not needed!
- Also an abstraction!

Logic as an abstraction

Logic as an abstraction

Logic is the subject that investigates *valid reasoning* while abstracting away from what is being reasoned about.

Logic as an abstraction

Logic as an abstraction

Logic is the subject that investigates *valid reasoning* while abstracting away from what is being reasoned about.

So Russell was right after all:

- Nobody knows what one is talking about. . . (A? B? C?)
- . . . nor what one is saying is true (what does "Greek" mean?)

Logic as an abstraction

Logic as an abstraction

Logic is the subject that investigates *valid reasoning* while abstracting away from what is being reasoned about.

So Russell was right after all:

- Nobody knows what one is talking about... (A? B? C?)
- ... nor what one is saying is true (what does "Greek" mean?)

And that is great, because it means that:

Computers can do this!

Logic as an abstraction

Logic as an abstraction

Logic is the subject that investigates *valid reasoning* while abstracting away from what is being reasoned about.

So Russell was right after all:

- Nobody knows what one is talking about... (A? B? C?)
- ...nor what one is saying is true (what does "Greek" mean?)

And that is great, because it means that:

Computers can do this!

Sure, cool, but why bother?

Outline

Models

- A *model* is a simplified representation of certain aspects of the real world.

Models

- A *model* is a simplified representation of certain aspects of the real world.
    - Also an abstraction

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting
  - communicating

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting
  - communicating
- Can be

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting
  - communicating

- Can be
  - Taxonomies (e.g. species, genus, family, etc. in biology)

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting
  - communicating

- Can be
  - Taxonomies (e.g. species, genus, family, etc. in biology)
  - Domain models, e.g. in UML

Models

- A *model* is a simplified representation of certain aspects of the real world.
  - Also an abstraction
- Made for
  - understanding
  - structuring
  - predicting
  - communicating

- Can be
  - Taxonomies (e.g. species, genus, family, etc. in biology)
  - Domain models, e.g. in UML
  - Numerical Models (Newtonian mechanics, Quantum mechanics)

Models in Computer Science

Models can be

- used to construct (parts of) software

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

- Check various properties of models

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

- Check various properties of models
- Check consistency between models

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

- Check various properties of models
- Check consistency between models
- Transform models from one language to another

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

- Check various properties of models
- Check consistency between models
- Transform models from one language to another
- Check that programs conform to models (verification)

Models in Computer Science

Models can be

- used to construct (parts of) software
  - Generate classes from UML diagrams
  - Generate code from UML sequence diagrams or state charts
- executed directly (sometimes)
  - Maude programs
  - Prolog programs
  - Models driving simulations
- used to check data
  - Database constraints = information model
  - XSD file = document model

So you also want algorithms to

- Check various properties of models
- Check consistency between models
- Transform models from one language to another
- Check that programs conform to models (verification)
- ...

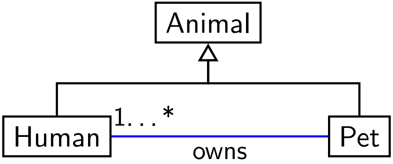
What is a Good Modelling Language

For a model that can be used by a computer…

- there has to be a 'language'
  - language says what is a model and what not
  - programs (and humans…) need to know what to expect
  - often defined by some grammar
  - UML, ER, ORM, OWL, SQL, Java, etc. are all languages
- the meaning of models should be very clear
  - otherwise, different implementations do different things
  - sometimes, 100s of pages of technical text (e.g. JLS)
  - sometimes meaning given by mathematical definitions

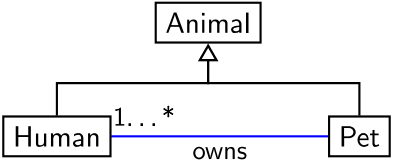
Models and Statements

Observation: much of the content of many models can be given as statements:



Models and Statements

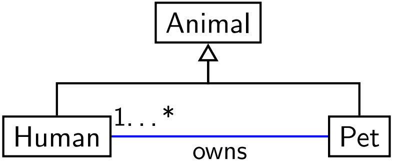
Observation: much of the content of many models can be given as statements:



- Every Human is an Animal.

Models and Statements

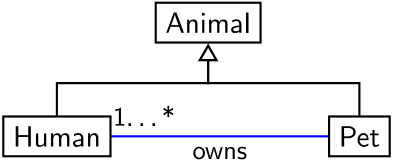
Observation: much of the content of many models can be given as statements:



- Every Human is an Animal.
- Every Pet is an Animal.

Models and Statements

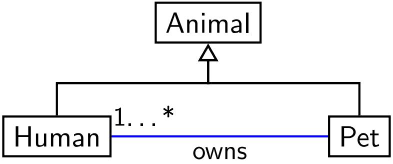
Observation: much of the content of many models can be given as statements:



- Every Human is an Animal.
- Every Pet is an Animal.
- Every Pet is owned by at least one Human.

Models and Statements

Observation: much of the content of many models can be given as statements:



- Every Human is an Animal.
- Every Pet is an Animal.
- Every Pet is owned by at least one Human.
- Everybody owning a Pet is a Human (?)

Modeling with Logic

- Logical languages are made for expressing statements

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

- $\forall x. \mathit{Human}(x) \rightarrow \mathit{Animal}(x)$

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

- $\forall x.Human(x) \to Animal(x)$
- $\forall x.Pet(x) \to Animal(x)$

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

- $\forall x.Human(x) \rightarrow Animal(x)$
- $\forall x.Pet(x) \rightarrow Animal(x)$
- $\forall x.Pet(x) \rightarrow \exists y.(Human(y) \wedge owns(y, x))$

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

- $\forall x. Human(x) \rightarrow Animal(x)$
- $\forall x. Pet(x) \rightarrow Animal(x)$
- $\forall x. Pet(x) \rightarrow \exists y. (Human(y) \land owns(y, x))$
- $\forall x, y. ((Pet(x) \land owns(y, x)) \rightarrow Human(y))$

Modeling with Logic

- Logical languages are made for expressing statements
  - They are more general than most other modeling languages
- Logical languages have a precise (mathematical) meaning
  - Implementation-independent by design
- Many things we do with models can be understood in terms of logical consequence

E.g.

- $\forall x.Human(x) \rightarrow Animal(x)$
- $\forall x.Pet(x) \rightarrow Animal(x)$
- $\forall x.Pet(x) \rightarrow \exists y.(Human(y) \wedge owns(y, x))$
- $\forall x, y.((Pet(x) \wedge owns(y, x)) \rightarrow Human(y))$

Logics are a very expressive and precise family of modelling languages

Outline

- What is Logic?
- Logic in Computer Science
- Three Ingredients
- Applications
- Course Information

Three Central Ingredients

- Syntax (i.e. the language)
- Semantics (i.e. the meaning)
- Calculus (i.e. method, algorithm, usually rules)

Syntax

Most logics have some kind of *formulas*.
The syntax says which strings of characters are formulas.

Syntax of Propositional Formulae

Propositional formulas are defined inductively as follows

- Every lower case letter (p, q, r,...) is a formula
- If A and B are formulae, then $\neg A$, $(A \wedge B)$, $(A \vee B)$ and $(A \rightarrow B)$ are formulae.

Inductively defined (remember IN1150): only what can be constructed using these rules is a formula.

$$p, \quad (p \wedge \neg p), \quad (p \rightarrow q) \vee (q \rightarrow p), \ldots$$

But not: $((p, \quad \neg \rightarrow q, \ldots$

Model Semantics

We usually define some kind of *interpretation* or *model* or *structure*...
Always the same idea:

- We don't know what we talk about (p, q, r, x, y, z)
- We don't know what is true (p or $\neg q$?)
- So we use a mathematical object that tells us what they mean and what is true or not

Interpretation

An interpretation is a function $\mathcal{I} : \mathit{Letters} \to \{T, F\}$ that assigns one of the truth values T or F to every lower case letter

Model Semantics (cont.)

Truth Value

The truth value of formulas $v_{\mathcal{I}}(A)$ is defined inductively by

- $v_{\mathcal{I}}(A) = \mathcal{I}(A)$ for letters A
- $v_{\mathcal{I}}(\neg A) = T$ if $v_{\mathcal{I}}(A) = F$ and
  $v_{\mathcal{I}}(\neg A) = F$ if $v_{\mathcal{I}}(A) = T$
- $v_{\mathcal{I}}(A \wedge B) = T$ if $v_{\mathcal{I}}(A) = T$ and $v_{\mathcal{I}}(B) = T$
  $v_{\mathcal{I}}(A \wedge B) = F$ otherwise
- ...

Entailment

Formula A *entails* formula B ($A \models B$) if for every $\mathcal{I}$ with $v_{\mathcal{I}}(A) = T$ it also holds that $v_{\mathcal{I}}(B) = T$.

Model Semantics: Take Aways

Model Semantics defines the meaning of logical formulas...

- i.e. truth/falsity in some interpretation/model/...
- relations between formulas like entailment, equivalence...

...by *mathematical definitions*.

- We assume that maths, set theory, etc. "work"
- We assume that people can read formulas, understand words like "and" or "not" or "otherwise," look up truth values in tables, etc.
- The definitions can often not be implemented directly
  - E.g. loop over infinitely many interpretations in 1st order logic

Calculi

- A calculus works on formulas, i.e. *syntax*
- Usually by *inference rules* saying how to *derive* new formulas

$$\frac{A \to B \qquad A}{B}$$

- Always with some machinery that says how to use the rules
- Can be used to check entailment etc. between formulas
- Can be implemented on a computer

Natural Deduction for Propositional Logic

- rules for $\land$ (conjunction)

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$A \qquad B$$

Natural Deduction for Propositional Logic

- rules for ∧ (conjunction)

$$\frac{A \qquad B}{A \land B} \land\text{-I}$$

Natural Deduction for Propositional Logic

- rules for ∧ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad A \wedge B$$

Natural Deduction for Propositional Logic

- rules for ∧ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\,\wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\,\wedge\text{-E}$$

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\,\wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\,\wedge\text{-E} \qquad\qquad A \wedge B$$

Natural Deduction for Propositional Logic

- rules for $\land$ (conjunction)

$$\frac{A \qquad B}{A \land B}\ \land\text{-I} \qquad\qquad \frac{A \land B}{A}\ \land\text{-E} \qquad\qquad \frac{A \land B}{B}\ \land\text{-E}$$

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\ \wedge\text{-E} \qquad\qquad \frac{A \wedge B}{B}\ \wedge\text{-E}$$

- rules for $\rightarrow$ (implication)

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\ \wedge\text{-E} \qquad\qquad \frac{A \wedge B}{B}\ \wedge\text{-E}$$

- rules for $\rightarrow$ (implication)

$$\begin{array}{c} [A]^n \\ \vdots \\ \vdots \\ B \end{array}$$

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\ \wedge\text{-E} \qquad\qquad \frac{A \wedge B}{B}\ \wedge\text{-E}$$

- rules for $\rightarrow$ (implication)

$$\begin{array}{c} [A]^n \\ \vdots \\ \vdots \\ \dfrac{B}{A \rightarrow B}\ \rightarrow\text{-I}^n \end{array}$$

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\ \wedge\text{-E} \qquad\qquad \frac{A \wedge B}{B}\ \wedge\text{-E}$$

- rules for $\rightarrow$ (implication)

$$\begin{array}{c}[A]^n\\ \vdots\\ \dfrac{B}{A \rightarrow B}\ \rightarrow\text{-I}^n\end{array} \qquad\qquad A \rightarrow B \qquad A$$

Natural Deduction for Propositional Logic

- rules for $\wedge$ (conjunction)

$$\frac{A \qquad B}{A \wedge B}\ \wedge\text{-I} \qquad\qquad \frac{A \wedge B}{A}\ \wedge\text{-E} \qquad\qquad \frac{A \wedge B}{B}\ \wedge\text{-E}$$

- rules for $\rightarrow$ (implication)

$$\frac{\begin{array}{c}[A]^n \\ \vdots \\ B\end{array}}{A \rightarrow B}\ \rightarrow\text{-I}^n \qquad\qquad \frac{A \rightarrow B \qquad A}{B}\ \rightarrow\text{-E}$$

Calculi: Take Aways

Calculi allow to determine

- *semantic properties* like equivalence, satisfiability etc.
- by *syntactic means*

. . . i.e. in ways that can be implemented

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Completeness

What is entailed can be derived

$$A \models B \implies A \vdash B$$

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Completeness

What is entailed can be derived

$$A \models B \implies A \vdash B$$

Two central properties; we will study how to prove them

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Completeness

What is entailed can be derived

$$A \models B \implies A \vdash B$$

Two central properties; we will study how to prove them

- for different logics

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Completeness

What is entailed can be derived

$$A \models B \implies A \vdash B$$

Two central properties; we will study how to prove them

- for different logics
- for different calculi

Semantics vs. Calculus

- Entailment ($A \models B$): semantic notion
- Derivability ($A \vdash B$): syntactic notion

Soundness

What can be derived is entailed

$$A \vdash B \implies A \models B$$

Completeness

What is entailed can be derived

$$A \models B \implies A \vdash B$$

Two central properties; we will study how to prove them

- for different logics
- for different calculi

Learn techniques to handle languages and their semantics

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics…

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics...
- ...how do we define model semantics!?

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics…
- …how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics. . .
- . . . how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

- More focus on manipulating proofs

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics. . .
- . . . how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

- More focus on manipulating proofs
- Known as the ‘proof theoretic approach’

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics...
- ...how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

- More focus on manipulating proofs
- Known as the 'proof theoretic approach'
- We concentrate on 'model theoretic approach'

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics. . .
- . . . how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

- More focus on manipulating proofs
- Known as the ‘proof theoretic approach’
- We concentrate on ‘model theoretic approach’
- As computer scientists, we take maths for granted.

Logic without Semantics?

- Much of mathematical logic stems from attempts at a formal foundation of mathematics itself
- End of 19th, beginning og 20th century
- Try to use logic to build mathematics from the bottom up
- But without mathematics...
- ...how do we define model semantics!?

Foundational mathematics considers logics without (model) semantics

- More focus on manipulating proofs
- Known as the 'proof theoretic approach'
- We concentrate on 'model theoretic approach'
- As computer scientists, we take maths for granted.
- Foundations are not (usually) our problem :-)

Outline

SAT

- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?

SAT

- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem

SAT

- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem
- NP-hard, i.e. takes time exponential in size of A

SAT

- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem
- NP-hard, i.e. takes time exponential in size of A
- Can often be done for very large problems, over 1M variables

SAT

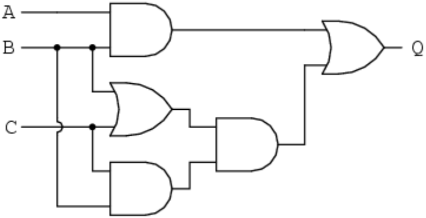
- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?
- https://en.wikipedia.org/wiki/Boolean_satisfiability_problem
- NP-hard, i.e. takes time exponential in size of A
- Can often be done for very large problems, over 1M variables
- Will learn more about how in a later lecture

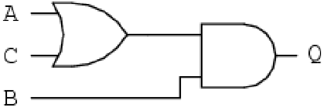
SAT

- SAT-solving: given a propositional formula A, is there an interpretation $\mathcal{I}$ such that $v_{\mathcal{I}}(A) = T$?
- `https://en.wikipedia.org/wiki/Boolean_satisfiability_problem`
- NP-hard, i.e. takes time exponential in size of A
- Can often be done for very large problems, over 1M variables
- Will learn more about how in a later lecture
- See here for a talk about applications `http://www.carstensinz.de/talks/RISC-2005.pdf`

SAT applications: circuit verification

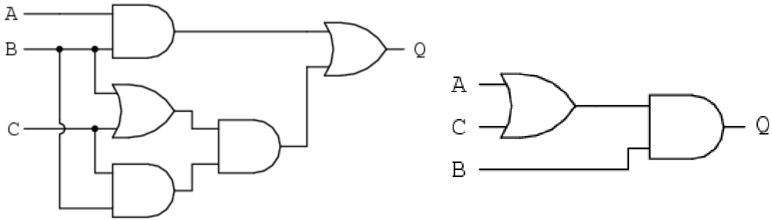
Are these two circuits the same?





SAT applications: circuit verification

Are these two circuits the same?

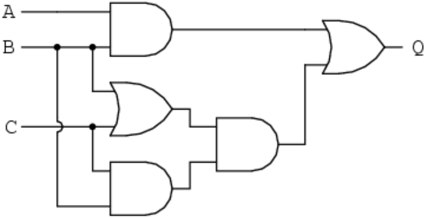


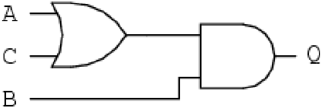
$(A \wedge B) \vee ((B \vee C) \wedge (B \wedge C))$ vs. $(A \vee C) \wedge B$

Logically equivalent?

SAT applications: circuit verification

Are these two circuits the same?





$(A \wedge B) \vee ((B \vee C) \wedge (B \wedge C))$ vs. $(A \vee C) \wedge B$

Logically equivalent?

Today, theorem provers are routinely used to check Boolean circuits

SAT applications: program verification

- A 32 bit int can be encoded as 32 boolean variables
- If SAT can handle 1M boolean variables, it can handle thousands of 32 bit words.
- Properties of programs (without loops) can be handled by SAT solvers
- Experimental, but works in many cases

SAT applications: puzzle solving

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

SAT applications: puzzle solving

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Use 4 bits to encode the number in each of the 81 squares ($\leq$ 324 bits)

SAT applications: puzzle solving

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Use 4 bits to encode the number in each of the 81 squares ($\leq$ 324 bits)
- Add axioms that ensure each number 1–9 occurs in each square, each row, each column

SAT applications: puzzle solving

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Use 4 bits to encode the number in each of the 81 squares ($\leq$ 324 bits)
- Add axioms that ensure each number 1–9 occurs in each square, each row, each column
- A satisfying interpretation is a solution of the puzzle

SAT applications: puzzle solving

5	3			7				
6			1	9	5			
	9	8					6	
8				6				3
4			8		3			1
7				2				6
	6					2	8	
			4	1	9			5
				8			7	9

- Use 4 bits to encode the number in each of the 81 squares ($\leq$ 324 bits)
- Add axioms that ensure each number 1–9 occurs in each square, each row, each column
- A satisfying interpretation is a solution of the puzzle

SAT: problems that require finding one of a large, fixed number of combinations, but checking is easy

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \land p(z, y))$

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \wedge p(z, y))$
- Undecidable in general

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \wedge p(z, y))$
- Undecidable in general
- More ‘brittle’ than SAT: small changes in formulation can make a big difference for a prover

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \wedge p(z, y))$
- Undecidable in general
- More 'brittle' than SAT: small changes in formulation can make a big difference for a prover
- Therefore fewer 'industrial' applications

First-order logic

- $\forall x \forall y \exists z (p(x, y) \to p(x, z) \land p(z, y))$
- Undecidable in general
- More 'brittle' than SAT: small changes in formulation can make a big difference for a prover
- Therefore fewer 'industrial' applications
- Can be used to formalise parts of mathematics (algebra)

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \land p(z, y))$
- Undecidable in general
- More ‘brittle’ than SAT: small changes in formulation can make a big difference for a prover
- Therefore fewer ‘industrial’ applications

- Can be used to formalise parts of mathematics (algebra)
- Add induction to reason about numbers and datatypes

First-order logic

- $\forall x \forall y \exists z (p(x, y) \rightarrow p(x, z) \wedge p(z, y))$
- Undecidable in general
- More 'brittle' than SAT: small changes in formulation can make a big difference for a prover
- Therefore fewer 'industrial' applications

- Can be used to formalise parts of mathematics (algebra)
- Add induction to reason about numbers and datatypes

- First-order theorem provers have been used to prove difficult open problems in (unintuitive parts of) mathematics

First-order logic

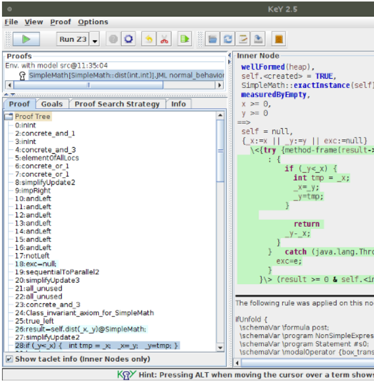
- $\forall x \forall y \exists z (p(x, y) \to p(x, z) \land p(z, y))$
- Undecidable in general
- More ‘brittle’ than SAT: small changes in formulation can make a big difference for a prover
- Therefore fewer ‘industrial’ applications

- Can be used to formalise parts of mathematics (algebra)
- Add induction to reason about numbers and datatypes

- First-order theorem provers have been used to prove difficult open problems in (unintuitive parts of) mathematics
- In combination with other techniques, first-order logic can used to reason about programs

The KeY tool

- `https://www.key-project.org/`
- Verify behaviour of Java programs
- Based on 1st-order logic
- Extended with program operators
  - $\langle Prog \rangle\ p$
    Prog terminates and p holds afterwards
- Based on a Sequent Calclus



The TimSort bug

Proving that Android's, Java's and Python's sorting algorithm is broken (and showing how to fix it)

February 24, 2015 Envisage Written by Stijn de Gouw. $s

Tim Peters developed the Timsort hybrid sorting algorithm in 2002. It is a clever combination of ideas from merge sort and insertion sort, and designed to perform well on real world data. TimSort was first developed for Python, but later ported to Java (where it appears as java.util.Collections.sort and java.util.Arrays.sort) by Joshua Bloch (the designer of Java Collections who also pointed out that most binary search algorithms were broken). TimSort is today used as the default sorting algorithm for Android SDK, Sun's JDK and OpenJDK. Given the popularity of these platforms this means that the number of computers, cloud services and mobile phones that use TimSort for sorting is well into the billions.

http://www.envisage-project.eu/proving-android-java-and-python-sorting-algorithm-is-broken-and-how-to-fix-it/

The World In Between

- Many logics are stronger than propositional logic, but still decidable.

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\ \text{hasChild.Professor}$

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\, \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\, \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,. . .

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\ \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,...
- E.g. Temporal Logic

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\ \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,. . .
- E.g. Temporal Logic
  - "Every request is eventually followed by an acknowledgement"

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - ProudMother $\equiv$ Person $\sqcap$ Female $\sqcap$ $\exists$ hasChild.Professor
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,...
- E.g. Temporal Logic
  - "Every request is eventually followed by an acknowledgement"
  - $\Box(\mathit{Req} \rightarrow \Diamond \mathit{Ack})$

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\, \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,...
- E.g. Temporal Logic
  - "Every request is eventually followed by an acknowledgement"
  - $\Box(Req \to \Diamond Ack)$
  - Can check properties of systems with hundreds of variables

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\ \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,...
- E.g. Temporal Logic
  - "Every request is eventually followed by an acknowledgement"
  - $\Box(Req \to \Diamond Ack)$
  - Can check properties of systems with hundreds of variables
  - Applications in dynamic circuit verification etc.

The World In Between

- Many logics are stronger than propositional logic, but still decidable.
- Often efficiently in practice
- E.g. Description Logics
  - "A ProudMother is a Person who is Female and has at least one child who is a Professor"
  - $\text{ProudMother} \equiv \text{Person} \sqcap \text{Female} \sqcap \exists\ \text{hasChild.Professor}$
  - Can define and reason about terminologies of up to 100.000s of concepts
  - Applications in semantic web, data integration,...
- E.g. Temporal Logic
  - "Every request is eventually followed by an acknowledgement"
  - $\Box(\mathit{Req} \rightarrow \Diamond \mathit{Ack})$
  - Can check properties of systems with hundreds of variables
  - Applications in dynamic circuit verification etc.
- Knowledge logics, probabilistic logics, alternating logics, belief logics, ...

Outline

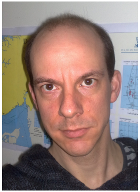
- What is Logic?
- Logic in Computer Science
- Three Ingredients
- Applications
- Course Information

When, Where, and Who

When and Where

- Lectures
  - Tuesdays 10:15–12:00 in OJD 2458, Postscript and/or
  - Thursdays 10:15–12:00 in KN Store Aud
  - When fewer students, move to OJD 3438, Caml
- Homepage: `https://www.uio.no/studier/emner/matnat/ifi/IN3070/index-eng.html`

Lecturer



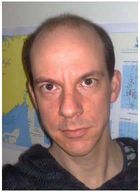
Martin Giese (`martingi@ifi.uio.no`)

When, Where, and Who

When and Where

- Lectures
  - Tuesdays 10:15–12:00 in OJD 2458, Postscript and/or
  - **Thursdays 10:15–12:00 in KN Store Aud**
  - When fewer students, move to OJD 3438, Caml
- Homepage: https://www.uio.no/studier/emner/matnat/ifi/IN3070/index-eng.html

Lecturer



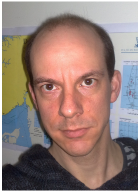
Martin Giese (martingi@ifi.uio.no)

When, Where, and Who

When and Where

- Lectures
  - Tuesdays 10:15–12:00 in OJD 2458, Postscript and/or
  - Thursdays 10:15–12:00 in KN Store Aud
  - **When fewer students, move to OJD 3438, Caml**
- Homepage: `https://www.uio.no/studier/emner/matnat/ifi/IN3070/index-eng.html`

Lecturer



Martin Giese (`martingi@ifi.uio.no`)

Exercises

Exercises

- Practical exercises every week,
- Thursdays 12:15–14:00 in OJD 3468 Fortress, from **27 August** and
- Tuesdays 10:15–12:00 in OJD 2458, Postscript, from 1 Sept.
- Exercises available on website well in advance. Come prepared!
- In general: part repetition of lectures, part exercises

Teacher

Ida Sandberg Motzfeldt
(`idasmot@ifi.uio.no`)

Exercises

Exercises

- Practical exercises every week,
- Thursdays 12:15–14:00 in OJD 3468 Fortress, from **27 August** and
- **Tuesdays 10:15–12:00 in OJD 2458, Postscript, from 1 Sept.**
- Exercises available on website well in advance. Come prepared!
- In general: part repetition of lectures, part exercises

Teacher

Ida Sandberg Motzfeldt
(idasmot@ifi.uio.no)

Mandatory Assignments

Assignments

- Two mandatory assignments (obliger)
- Will be in October/November
- Corrected by teacher.
- Pass/Fail
- Must have passed all assignments in order to attend exam
- For IN4070 (MSc version): one extra question in each oblig

Prover Hacking

- One of your assignments will be to program a simple prover.

Prover Hacking

- One of your assignments will be to program a simple prover.
- Doesn't need to be very powerful to get 'pass.'

Prover Hacking

- One of your assignments will be to program a simple prover.
- Doesn't need to be very powerful to get 'pass.'
- But hacking is fun, right?

Prover Hacking – Competition?

- One of your assignments will be to program a simple prover.
- Doesn't need to be very powerful to get 'pass.'
- But hacking is fun, right?
- May organise a little prover competition at the end of the semester.

Prover Hacking – Competition?

- One of your assignments will be to program a simple prover.
- Doesn't need to be very powerful to get 'pass.'
- But hacking is fun, right?
- May organise a little prover competition at the end of the semester.
- Strictly for fun, no influence on grade.

Padlet

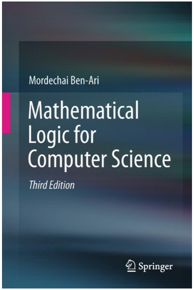
https://uio.padlet.org/martingi/8swc2uezt4sy2nsk

Exam

- Four hours written home exam
  - In case of few students, might be oral exam instead
- Same exam for IN3070 and IN4070
- Grades A–F
- Probably 4 December – Check semester page!
- Unsure, due to Corona

Textbook

- Mordechai Ben-Ari
  Mathematical Logic for Computer Science
  3rd edition, Springer, 2012.



- only chapters 1–4 and 6–12;
  not part of the curriculum:
  chapter 5 (binary decision diagrams) and
  chapters 13–16 (temporal logic, verification of programs)
- download for free (within the UiO network) from Springer's website at
  `http://www.springer.com/gp/book/9781447141280`

Next weeks. . .

- Propositional Logic
- Tableaux/Sequent calculi for propositional Logic
- Soundness and Completeness
- Resolution calculus for propositional Logic
- Soundness and Completeness
- First-order logic
- Tableaux/Sequent calculi and resolution for 1st order logic
- Soundness and Completeness for those calculi