Fundamentals of Programming Languages I

Introduction and Logics

Guoqiang Li

School of Software, Shanghai Jiao Tong University

Instructor and Teaching Assistants

- Guoqiang LI

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Prog17/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Prog17/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
- TA:
  - Yuwei WANG: wangyuwei95 (AT) qq (DOT) com

Instructor and Teaching Assistants

- Guoqiang LI
  - Homepage: http://basics.sjtu.edu.cn/~liguoqiang
  - Course page: http://basics.sjtu.edu.cn/~liguoqiang/teaching/Prog17/index.htm
  - Email: li.g@outlook.com
  - Office: Rm. 1212, Building of Software
  - Phone: 3420-4167
- TA:
  - Yuwei WANG: wangyuwei95 (AT) qq (DOT) com
- Office hour: Tue. 14:00-17:00 @ Software Building 3203

What does the lecture aim for?

Similar Lectures I

Fundamentals of Programming Languages by University of Colorado Boulder

http://www.cs.colorado.edu/~bec/courses/csci5535-f13/

Similar Lectures I

Fundamentals of Programming Languages by University of Colorado Boulder

http://www.cs.colorado.edu/~bec/courses/csci5535-f13/

- 2010 Spring Programming semantics
- 2013 Fall Programming analysis and verification

Similar Lectures II

Principles of Programming Languages by University of Oxford

http://www.cs.ox.ac.uk/teaching/courses/2017-2018/principles/

Foundations of Programming Languages by CMU

www.cs.cmu.edu/˜rjsimmon/15312-s14/schedule.html

Theory of Programming Languages by ECNU

basics.sjtu.edu.cn/˜yuxin/teaching/Semantics/sem.html

Similar Lectures II

Principles of Programming Languages by University of Oxford

http://www.cs.ox.ac.uk/teaching/courses/2017-2018/principles/

Foundations of Programming Languages by CMU

www.cs.cmu.edu/~rjsimmon/15312-s14/schedule.html

Theory of Programming Languages by ECNU

basics.sjtu.edu.cn/~yuxin/teaching/Semantics/sem.html

Similar Lectures III

Fundamentals of Programming Analysis by MIT

ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-820-fundamentals-of-program-analysis-fall-2015/lecture-notes/

Principles of Programming Languages by Boston University

http://www.cs.bu.edu/~hwxi/academic/courses/CS520/Fall15

Similar Lectures III

Fundamentals of Programming Analysis by MIT

ocw.mit.edu/courses/electrical-engineering-and-computer-science/6-820-fundamentals-of-program-analysis-fall-2015/lecture-notes/

Principles of Programming Languages by Boston University

http://www.cs.bu.edu/˜hwxi/academic/courses/CS520/Fall15

Programming Analysis and Verification

Similar Lectures IV

Theory of Programming Languages by CMU

www.cs.cmu.edu/ aldrich/courses/15-819O-13sp

Introduction to Programming Languages Theory by Standford

https://courseware.stanford.edu/pg/courses/lectures/261141

Theory of Programming Languages by SJTU

http://basics.sjtu.edu.cn/˜xiaojuan/tapl2016/index.html

Similar Lectures IV

Theory of Programming Languages by CMU

www.cs.cmu.edu/ aldrich/courses/15-819O-13sp

Introduction to Programming Languages Theory by Standford

https://courseware.stanford.edu/pg/courses/lectures/261141

Theory of Programming Languages by SJTU

http://basics.sjtu.edu.cn/˜xiaojuan/tapl2016/index.html

Fundamental Requirements

- Program Verification and Analysis

Fundamental Requirements

- Program Verification and Analysis
  - Propositional logic, predicate logic etc.
  - Automata theory, DFA, NFA, PDS, PN etc.
  - Algorithm.

Fundamental Requirements

- Program Verification and Analysis
  - Propositional logic, predicate logic etc.
  - Automata theory, DFA, NFA, PDS, PN etc.
  - Algorithm.
- Program Semantics

Fundamental Requirements

- Program Verification and Analysis
  - Propositional logic, predicate logic etc.
  - Automata theory, DFA, NFA, PDS, PN etc.
  - Algorithm.
- Program Semantics
  - Set theory.
  - Algebra theory, group, ring, domain etc.
  - category theory, maybe...

Fundamental Requirements

- Program Verification and Analysis
  - Propositional logic, predicate logic etc.
  - Automata theory, DFA, NFA, PDS, PN etc.
  - Algorithm.
- Program Semantics
  - Set theory.
  - Algebra theory, group, ring, domain etc.
  - category theory, maybe...
- Types and Programming Languages

Fundamental Requirements

- Program Verification and Analysis
  - Propositional logic, predicate logic etc.
  - Automata theory, DFA, NFA, PDS, PN etc.
  - Algorithm.
- Program Semantics
  - Set theory.
  - Algebra theory, group, ring, domain etc.
  - category theory, maybe...
- Types and Programming Languages
  - Logic
  - Computability theory
  - Lambda calculus theory...

Fundamental of Fundamental

Several theories in theoretical computer science are given, which is a minimal requirement and self-contained in this lecture.

Fundamental of Fundamental

Several theories in theoretical computer science are given, which is a minimal requirement and self-contained in this lecture.

All of three directions are taught, which only include very fundamental part,

Fundamental of Fundamental

Several theories in theoretical computer science are given, which is a minimal requirement and self-contained in this lecture.

All of three directions are taught, which only include very fundamental part, if time permitted.

Fundamental of Fundamental

Several theories in theoretical computer science are given, which is a minimal requirement and self-contained in this lecture.

All of three directions are taught, which only include very fundamental part, if time permitted.

As simple as possible,

Fundamental of Fundamental

Several theories in theoretical computer science are given, which is a minimal requirement and self-contained in this lecture.

All of three directions are taught, which only include very fundamental part, if time permitted.

As simple as possible, although it is very theoretical.

Lecture Agenda

- Introduction and logic basics (1 lecture)
- Formal basics (3 lectures)
- Programming verification (2 or 3 lectures)
- Exercise I. (1 lecture)
- Programming semantics (2 lectures)
- Basic functional programming (3 lectures)
- Exercise II. (1 lecture)
- Conclusion and wrap up (1 lecture)

Lecture Agenda

- Introduction and logic basics (1 lecture)
- Formal basics (3 lectures)
  - Model checking
  - Finite and Büchi automata
  - LTL model checking
- Programming verification (2 or 3 lectures)
- Exercise I. (1 lecture)
- Programming semantics (2 lectures)
- Basic functional programming (3 lectures)
- Exercise II. (1 lecture)
- Conclusion and wrap up (1 lecture)

Lecture Agenda

- Introduction and logic basics (1 lecture)
- Formal basics (3 lectures)
- Programming verification (2 or 3 lectures)
    - Abstract interpretation
    - Pushdown automata and interprocedural programs
    - Petri Net and concurrent programs
- Exercise I. (1 lecture)
- Programming semantics (2 lectures)
- Basic functional programming (3 lectures)
- Exercise II. (1 lecture)
- Conclusion and wrap up (1 lecture)

Lecture Agenda

- Introduction and logic basics (1 lecture)
- Formal basics (3 lectures)
- Programming verification (2 or 3 lectures)
- Exercise I. (1 lecture)
- Programming semantics (2 lectures)
  - Denotational semantics
  - Operational semantics
  - Axiomatic semantics
- Basic functional programming (3 lectures)
- Exercise II. (1 lecture)
- Conclusion and wrap up (1 lecture)

Lecture Agenda

- Introduction and logic basics (1 lecture)
- Formal basics (3 lectures)
- Programming verification (2 or 3 lectures)
- Exercise I. (1 lecture)
- Programming semantics (2 lectures)
- Basic functional programming (3 lectures)
    - Lambda calculus
    - Simple types
    - Functional programming
- Exercise II. (1 lecture)
- Conclusion and wrap up (1 lecture)

References

No particular textbook that can cover all the parts. Here are three Reference books:

Edmund M. Clarke Jr., Orna Grumberg, Doron A. Peled. Model Checking. MIT Press, 1999

Glynn Winskel. Formal Semantics of Programming Languages: An Introduction. MIT Press, 1993

Benjamin C. Pierce. Types and Programming Languages. MIT Press, 2002

References

No particular textbook that can cover all the parts. Here are three Reference books:

Edmund M. Clarke Jr., Orna Grumberg, Doron A. Peled. Model Checking. MIT Press, 1999

Glynn Winskel. Formal Semantics of Programming Languages: An Introduction. MIT Press, 1993

Benjamin C. Pierce. Types and Programming Languages. MIT Press, 2002

+ Several famous papers

+ Lecture notes shared in the course webpage.

Scoring Policy

- 10% Attendance.
- 20% Homework.
- 70% Final exam.

Scoring Policy

- 10% Attendance.
- 20% Homework.
    - Four assignments.
- 70% Final exam.

Scoring Policy

- 10% Attendance.
- 20% Homework.
  - Four assignments.
  - Each one is 5pts.
- 70% Final exam.

Scoring Policy

- 10% Attendance.
- 20% Homework.
  - Four assignments.
  - Each one is 5pts.
  - Work out individually.
- 70% Final exam.

Scoring Policy

- 10% Attendance.
- 20% Homework.
    - Four assignments.
    - Each one is 5pts.
    - Work out individually.
    - Each assignment will be evaluated by A, B, C, D, F (Excellent(5), Good(5), Fair(4), Delay(3), Fail(0))
- 70% Final exam.

Scoring Policy

- 10% Attendance.
- 20% Homework.
  - Four assignments.
  - Each one is 5pts.
  - Work out individually.
  - Each assignment will be evaluated by A, B, C, D, F (Excellent(5), Good(5), Fair(4), Delay(3), Fail(0))
- 70% Final exam.
  - Maybe replaced by report, if the condition is satisfied!

Any Questions?

Logic Basics

Brief Historical Notes on Logic

Historical View

- Philosophical Logic
  - 500 BC to 19th Century
- Symbolic Logic
  - Mid to late 19th Century
- Mathematical Logic
  - Late 19th to mid 20th Century
- Logic in Computer Science

Philosophical Logic

Philosophical Logic

500 B.C - 19th Century

Philosophical Logic

500 B.C - 19th Century

Logic dealt with arguments in the natural language used by humans.

Philosophical Logic

500 B.C - 19th Century

Logic dealt with arguments in the natural language used by humans.

Example:

- All men are mortal.
- Socrates is a man.
- Therefore, Socrates is mortal.

Philosophical Logic

Philosophical Logic

Natural languages are very ambiguous.

Philosophical Logic

Natural languages are very ambiguous.

- Eric does not believe that Mary can pass any test.
  - does not believe that she can pass some test, or
  - does not believe that she can pass all tests
- I only borrowed your car.
  - And not 'borrowed and used', or
  - And not 'car and coat'
- Tom hates Jim and he likes Mary.
  - Tom likes Mary, or
  - Jim likes Mary

Philosophical Logic

Natural languages are very ambiguous.

- Eric does not believe that Mary can pass any test.
  - does not believe that she can pass some test, or
  - does not believe that she can pass all tests
- I only borrowed your car.
  - And not ‘borrowed and used’, or
  - And not ‘car and coat’
- Tom hates Jim and he likes Mary.
  - Tom likes Mary, or
  - Jim likes Mary

It led to many paradoxes.

Philosophical Logic

Natural languages are very ambiguous.

- Eric does not believe that Mary can pass any test.
  - does not believe that she can pass some test, or
  - does not believe that she can pass all tests
- I only borrowed your car.
  - And not ‘borrowed and used’, or
  - And not ‘car and coat’
- Tom hates Jim and he likes Mary.
  - Tom likes Mary, or
  - Jim likes Mary

It led to many paradoxes.

- “This sentence is a lie.”(The Liar’s Paradox)

Sophism

…Sophism generally refers to a particularly confusing, illogical and/or insincere argument used by someone to make a point, or, perhaps, not to make a point.

Sophistry refers to […] rhetoric that is designed to appeal to the listener on grounds other than the strict logical cogency of the statements being made.

The Sophist's Paradox

A Sophist is sued for his tuition by the school that educated him. He argues that he must win, since, if he loses, the school didn't educate him well enough, and doesn't deserve the money.

The Sophist's Paradox

A Sophist is sued for his tuition by the school that educated him. He argues that he must win, since, if he loses, the school didn't educate him well enough, and doesn't deserve the money.

The school argues that he must lose, since, if he wins, he was educated well enough, and therefore should pay for it.

Logic in Computer Science

Logic has a profound impact on computer science. Some examples:

- Propositional logic - the foundation of computers and circuitry
- Databases - query languages
- Programming languages (e.g. prolog)
- Design Validation and verification
- AI (e.g. inference systems)
- …

Logic in Computer Science

Propositional Logic

First Order Logic

Higher Order Logic

Temporal Logic

…

Propositional Logic: Syntax

Propositional Logic

Propositional Logic

A proposition: a sentence that can be either true or false.

Propositional Logic

A proposition: a sentence that can be either true or false.

Propositions:

- x is greater than y
- Noam wrote this letter

Propositional Logic: Syntax

Propositional Logic: Syntax

The symbols of the language:

- Propositional symbols (*Prop*): $A, B, C, \ldots$
- Connectives:
  - $\wedge$ and
  - $\vee$ or
  - $\neg$ not
  - $\rightarrow$ implies
  - $\leftrightarrow$ equivalent to
  - $\oplus$ xor (different than)
  - $\bot, \top$ False, True
- Parenthesis: $(,)$.

Propositional Logic: Syntax

The symbols of the language:

- Propositional symbols ($Prop$): $A, B, C, \ldots$
- Connectives:
  - $\wedge$ and
  - $\vee$ or
  - $\neg$ not
  - $\rightarrow$ implies
  - $\leftrightarrow$ equivalent to
  - $\oplus$ xor (different than)
  - $\perp, \top$ False, True
- Parenthesis: $(,)$.

Q1: How many different binary symbols can we define?

Propositional Logic: Syntax

The symbols of the language:

- Propositional symbols ($Prop$): $A, B, C, \ldots$
- Connectives:
  - $\wedge$ and
  - $\vee$ or
  - $\neg$ not
  - $\rightarrow$ implies
  - $\leftrightarrow$ equivalent to
  - $\oplus$ xor (different than)
  - $\perp, \top$ False, True
- Parenthesis: $($, $)$.

Q1: How many different binary symbols can we define?

Q2: What is the minimal number of such symbols?

Formulas

Grammar of well-formed propositional formulas

$$Formula := prop \mid \neg(Formula) \mid (Formula \circ Formula)$$

where $prop \in Prop$ and $\circ$ is one of the binary relations.

Formulas

Examples of well-formed formulas:

- $(\neg A)$
- $(\neg(\neg A))$
- $(A \wedge (B \wedge C))$
- $(A \rightarrow (B \rightarrow C))$

Correct expressions of Propositional Logic are full of unnecessary parenthesis.

Formulas: Abbreviations

We write

$$A \circ B \circ C \circ \ldots$$

Formulas: Abbreviations

We write

$$A \circ B \circ C \circ \ldots$$

in place of

$$(A \circ (B \circ (C \circ \ldots)))$$

Formulas: Abbreviations

We write

$$A \circ B \circ C \circ \ldots$$

in place of

$$(A \circ (B \circ (C \circ \ldots)))$$

Thus, we write

$$A \wedge B \wedge C, \qquad A \to B \to C, \ldots$$

Formulas: Abbreviations

We write

$$A \circ B \circ C \circ \ldots$$

in place of

$$(A \circ (B \circ (C \circ \ldots)))$$

Thus, we write

$$A \wedge B \wedge C, \qquad A \to B \to C, \ldots$$

in place of

$$(A \wedge (B \wedge C)), \qquad (A \to (B \to C)), \ldots$$

Formulas: Abbreviations

We omit parenthesis whenever we may restore them through operator precedence:

Formulas: Abbreviations

We omit parenthesis whenever we may restore them through operator precedence:

$\neg$ binds more strictly than $\wedge$, $\vee$, and $\wedge$, $\vee$ bind more strictly than $\rightarrow$, $\leftrightarrow$.

Formulas: Abbreviations

We omit parenthesis whenever we may restore them through operator precedence:

$\neg$ binds more strictly than $\wedge$, $\vee$, and $\wedge$, $\vee$ bind more strictly than $\rightarrow$, $\leftrightarrow$.

Thus, we write:

- $\neg\neg A$ for $(\neg(\neg A))$,
- $\neg A \wedge B$ for $((\neg A) \wedge B)$
- $A \wedge B \rightarrow C$ for $((A \wedge B) \rightarrow C)$
- …

Propositional Logic: Semantics

Propositional Logic: Semantics

Truth tables define the semantics (=meaning) of the operators

Convention: $0 = \mathit{false}, 1 = \mathit{true}$

A	B	$A \wedge B$	$A \vee B$	$A \rightarrow B$
0	0	0	0	1
0	1	0	1	1
1	0	0	1	0
1	1	1	1	1

Propositional Logic: Semantics

Truth tables define the semantics (=meaning) of the operators

Convention: $0 = \mathit{false}, 1 = \mathit{true}$

A	B	$\neg A$	$A \leftrightarrow B$	$A \oplus B$
0	0	1	1	0
0	1	1	0	1
1	0	0	0	1
1	1	0	1	0

Back to Q1

Q1: How many binary operators can we define that have different semantic definition?

Back to Q1

Q1: How many binary operators can we define that have different semantic definition?

A: 16

Satisfiability and Validity

Assignments

Definition: A truth-values assignment, α, is an element of 2^{Prop} (i.e., $\alpha \in 2^{Prop}$).

In other words, α is a subset of the variables that are assigned true.

Equivalently, we can see α as a mapping from variables to truth values:

$$\alpha : Prop \mapsto \{0, 1\}$$

Example: $\alpha = \{A \mapsto 0, B \mapsto 1, \ldots\}$

Satisfaction Relation ($\models$): Intuition

An assignment can either satisfy or not satisfy a given formula.

$\alpha \models \phi$ means

- α satisfies ϕ or
- ϕ holds at α or
- α is a model of ϕ

We will first see an example.

Then we will define these notions formally.

Example

Let $\phi = (A \vee (B \to C))$

Example

Let $\phi = (A \vee (B \rightarrow C))$

Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Example

Let $\phi = (A \vee (B \rightarrow C))$

Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Q: Does α satisfy ϕ ($\alpha \models \phi$?)

Example

Let $\phi = (A \vee (B \rightarrow C))$

Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Q: Does α satisfy ϕ ($\alpha \models \phi$?)

A: $(0 \vee (0 \rightarrow 1)) = (0 \vee 1) = 1$

Example

Let $\phi = (A \vee (B \rightarrow C))$

Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Q: Does α satisfy ϕ ($\alpha \models \phi$?)

A: $(0 \vee (0 \rightarrow 1)) = (0 \vee 1) = 1$
Hence, $\alpha \models \phi$.

Example

Let $\phi = (A \vee (B \rightarrow C))$

Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Q: Does α satisfy ϕ ($\alpha \models \phi$?)

A: $(0 \vee (0 \rightarrow 1)) = (0 \vee 1) = 1$
Hence, $\alpha \models \phi$.

Let us now formalize an evaluation process.

Satisfaction Relation ($\models$): Formalities

$\models$ is a relation: $\models \subseteq (2^{Prop} \times \mathit{Formula})$

Satisfaction Relation ($\models$): Formalities

$\models$ is a relation: $\models \subseteq (2^{Prop} \times \mathit{Formula})$

Examples:

- $(\{A\}, A \vee B)$: the assignment $\alpha = \{A\}$ satisfies $A \vee B$
- $(\{A, B\}, A \wedge B)$

Satisfaction Relation ($\models$): Formalities

$\models$ is a relation: $\models \subseteq (2^{Prop} \times \mathit{Formula})$

Examples:

- $(\{A\}, A \vee B)$: the assignment $\alpha = \{A\}$ satisfies $A \vee B$
- $(\{A, B\}, A \wedge B)$

Alternatively: $\models \subseteq (\{0, 1\}^{Prop} \times \mathit{Formula})$

Satisfaction Relation ($\models$): Formalities

$\models$ is a relation: $\models \subseteq (2^{Prop} \times Formula)$

Examples:

- $(\{A\}, A \vee B)$: the assignment $\alpha = \{A\}$ satisfies $A \vee B$
- $(\{A, B\}, A \wedge B)$

Alternatively: $\models \subseteq (\{0, 1\}^{Prop} \times Formula)$

Examples:

- $(01, A \vee B)$: the assignment $\alpha = \{A \mapsto 0, B \mapsto 1\}$ satisfies $A \vee B$
- $(11, A \wedge B)$

Satisfaction Relation ($\models$): Formalities

$\models$ is defined recursively:

- $\alpha \models A$ if $\alpha(A) = \mathit{true}$
- $\alpha \models \neg\varphi$ if $\alpha \not\models \varphi$
- $\alpha \models \varphi_1 \wedge \varphi_2$ if $\alpha \models \varphi_1$ and $\alpha \models \varphi_2$
- $\alpha \models \varphi_1 \vee \varphi_2$ if $\alpha \models \varphi_1$ or $\alpha \models \varphi_2$
- $\alpha \models \varphi_1 \rightarrow \varphi_2$ if $\alpha \models \varphi_1$ implies $\alpha \models \varphi_2$
- $\alpha \models \varphi_1 \leftrightarrow \varphi_2$ if $\alpha \models \varphi_1$ iff $\alpha \models \varphi_2$

From Definition to an Evaluation Algorithm

Truth Evaluation Problem:
Given $\varphi \in \mathit{Formula}$ and $\alpha \in 2^{AP}(\varphi)$, does $\alpha \models \varphi$?

From Definition to an Evaluation Algorithm

Truth Evaluation Problem:
Given $\varphi \in \mathit{Formula}$ and $\alpha \in 2^{AP}(\varphi)$, does $\alpha \models \varphi$?

$\texttt{Eval}(\varphi, \alpha)$

if $\varphi \equiv A$ **then** $\texttt{return}\ \alpha(A)$;
if $\varphi \equiv \neg\phi$ **then** $\texttt{return}\ \neg\ \texttt{Eval}(\phi, \alpha)$;
if $\varphi \equiv \psi \circ \phi$ **then**
$\texttt{return Eval}(\psi, \alpha) \circ \texttt{Eval}(\phi, \alpha)$;

From Definition to an Evaluation Algorithm

Truth Evaluation Problem:
Given $\varphi \in \mathit{Formula}$ and $\alpha \in 2^{AP}(\varphi)$, does $\alpha \models \varphi$?

`Eval` (φ, α)

if $\varphi \equiv A$ **then** `return` $\alpha(A)$;
if $\varphi \equiv \neg\phi$ **then** `return` $\neg$ `Eval` (ϕ, α);
if $\varphi \equiv \psi \circ \phi$ **then**
`return` `Eval` $(\psi, \alpha) \circ$ `Eval` (ϕ, α);

`Eval` uses polynomial time and space.

Nothing More Than What We Already Know

Recall the Example:

- Let $\phi = (A \vee (B \to C))$
- Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

Nothing More Than What We Already Know

Recall the Example:

- Let $\phi = (A \vee (B \rightarrow C))$
- Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

$\texttt{Eval}(\phi, \alpha) = \texttt{Eval}(A, \alpha) \vee \texttt{Eval}(B \rightarrow C, \alpha) =$
$0 \vee \texttt{Eval}(B, \alpha) \rightarrow \texttt{Eval}(C, \alpha) = 0 \vee (0 \rightarrow 1) = 0 \vee 1 = 1$

Nothing More Than What We Already Know

Recall the Example:

- Let $\phi = (A \vee (B \rightarrow C))$
- Let $\alpha = \{A \mapsto 0, B \mapsto 0, C \mapsto 1\}$

$\texttt{Eval}(\phi, \alpha) = \texttt{Eval}(A, \alpha) \vee \texttt{Eval}(B \rightarrow C, \alpha) =$
$0 \vee \texttt{Eval}(B, \alpha) \rightarrow \texttt{Eval}(C, \alpha) = 0 \vee (0 \rightarrow 1) = 0 \vee 1 = 1$

Hence, $\alpha \models \phi$.

Extending Truth Table

p	q	$(p \to (q \to p))$	$(p \wedge \neg p)$	$(p \vee \neg q)$
0	0	1	0	1
0	1	1	0	0
1	0	1	0	1
1	1	1	0	1

Extending Truth Table

p	q	r	$(p \rightarrow (q \rightarrow \neg r)$
0	0	0	
0	0	1	
0	1	0	
0	1	1	
1	0	0	
1	0	1	
1	0	0	
1	1	1	

Extending Truth Table

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

Set of Assignment

Intuition: a formula specifies a set of truth assignments.

Function models: $models : Formula \mapsto 2^{2^{Prop}}$
(a formula $\mapsto$ set of satisfying assignments)

Recursive definition:

- $models(A) = \{\alpha | \alpha(A) = 1\}, A \in Prop$
- $models(\neg\varphi) = 2^{Prop} - models(\varphi)$
- $models(\varphi_1 \wedge \varphi_2) = models(\varphi_1) \cap models(\varphi_2)$
- $models(\varphi_1 \vee \varphi_2) = models(\varphi_1) \cup models(\varphi_2)$
- $models(\varphi_1 \rightarrow \varphi_2) = (2^{Prop} - models(\varphi_1) \cup models(\varphi_2)$

Example

$models(A \vee B) = \{\{10\}, \{01\}, \{11\}\}$

This is compatible with the recursive definition:

$$models(A \vee B) = models(A) \cup models(B) =$$
$$\{\{10\}, \{11\}\} \cup \{\{01\}, \{11\}\} =$$
$$\{\{10\}, \{01\}, \{11\}\}$$

Theorem

Let $\varphi \in \mathit{Formula}$ and $\alpha \in 2^{\mathit{Prop}}$, then the following statements are equivalent:

- $\alpha \models \varphi$
- $\alpha \in \mathit{models}(\varphi)$

Projected Assignment

$AP(\varphi)$: the Atomic Propositions in φ.

Clearly $AP(\varphi) \subseteq Prop$.

Let $\alpha_1, \alpha_2 \in 2^{Prop}$, $\in$ *Formula*.

Lemma: if $\alpha_1 \mid_{AP(\varphi)} = \alpha_2 \mid_{AP(\varphi)}$, then

$$\alpha_1 \models \varphi \text{ iff } \alpha_2 \models \varphi$$

Corollary: $\alpha \models \varphi$ iff $\alpha \mid_{AP(\varphi)} \models \varphi$

We will assume, for simplicity, that $Prop = AP(\varphi)$.

Extension of $\models$ to Assignment Sets

Let $\varphi \in \mathit{Formula}$

Let T be a set of assignments, i.e., $T \subseteq 2^{2^{\mathit{Prop}}}$

Definition. $T \models \varphi$ if $T \subseteq \mathit{models}(\varphi)$

i.e., $\models \subseteq 2^{2^{\mathit{Prop}}} \times \mathit{Formula}$

Extension of $\models$ to Formulas

$\models \subseteq 2^{Formula} \times 2^{Formula}$

Definition. Let Γ_1, Γ_2 be prop. formulas.

$$\Gamma_1 \models \Gamma_2$$

iff $models(\Gamma_1) \subseteq models(\Gamma_2)$

iff for all $\alpha \in 2^{Prop}$ if $\alpha \models \Gamma_1$ then $\alpha \models \Gamma_2$

Examples:

$$x_1 \wedge x_2 \models x_1 \vee x_2$$
$$x_1 \wedge x_2 \models x_2 \vee x_3$$

Classification of Formulas

A formula φ is called valid if $\mathit{models}(\varphi) = 2^{\mathit{Prop}}$.
(also called a tautology).

A formula φ is called satisfiable if $\mathit{models}(\varphi) \neq \emptyset$.

A formula φ is called unsatisfiable if $\mathit{models}(\varphi) = \emptyset$
(also called a contradiction).

Characteristics of Formulas

A formula φ is valid iff $\neg\varphi$ is unsatisfiable.

φ is satisfiable iff $\neg\varphi$ is not valid.

Characteristics of Formulas

We can write

$\models \varphi$ when φ is valid.

$\not\models \varphi$ when φ is not valid.

$\not\models \neg\varphi$ when φ is satisfiable.

$\models \neg\varphi$ when φ is unsatisfiable

Examples

$(p \wedge q) \rightarrow (p \vee q)$	is	valid
$(p \vee q) \rightarrow p$	is	satisfiable
$(p \wedge q) \wedge \neg p$	is	unsatisfiable

Equivalences

$\models A \wedge 1 \leftrightarrow A$

$\models A \wedge 0 \leftrightarrow 0$

$\models \neg\neg A \leftrightarrow A$

$\models A \wedge (B \vee C) \leftrightarrow (A \wedge B) \vee (A \wedge C)$

$\models \neg(A \wedge B) \leftrightarrow (\neg A \vee \neg B)$

$\models \neg(A \vee B) \leftrightarrow (\neg A \wedge \neg B)$

Minimal Set of Binary Operators

Recall the question: what is the minimal set of operators necessary?

A: Through such equivalences all Boolean operators can be written with a single operator ($\oplus$).

Indeed, typically industrial circuits only use one type of logical gate.

We'll see how two are enough: $\neg$ and $\wedge$

- Or: $\models (A \vee B) \leftrightarrow \neg(\neg A \wedge \neg B)$
- Implies: $\models (A \rightarrow B) \leftrightarrow (\neg A \vee B)$
- Equivalence: $\models (A \leftrightarrow B) \leftrightarrow (A \rightarrow B) \wedge (B \rightarrow A)$
- …

Decision Problem

The decision problem:

Given a propositional formula ϕ, is ϕ satisfiable?

An algorithm that always terminates with a correct answer to this problem is called a decision procedure for propositional logic.

Normal Forms

Definitions

A literal is either an atom or a negation of an atom.

Let $\phi = \neg(A \vee \neg B)$. Then:

- Atoms: $AP(\phi) = \{A, B\}$
- Literals: $lit(\phi) = \{A, \neg B\}$

Equivalent formulas can have different literals

- $\phi = \neg(A \vee \neg B) = \neg A \wedge B$
- Now $lit(\phi) = \{\neg A, B\}$

Definitions

A term is a conjunction of literals

- Example: $(A \wedge \neg B \wedge C)$

A clause is a disjunction of literals

- Example: $(A \vee \neg B \vee C)$

Negation Normal Form (NNF)

A formula is said to be in Negation Normal Form (NNF) if it only contains $\neg$, $\wedge$, $\vee$ connectives and only atoms can be negated.

Examples:

- $\neg(A \vee \neg B)$ is not in NNF
- $\neg A \wedge B$ is in NNF

Coverting to NNF

Every formula can be converted to NNF in linear time:

- Eliminate all connectives other than $\wedge, \vee, \neg$
- Use De Morgan and double-negation rules to push negations to the right

Example: $\neg(A \rightarrow \neg B)$

- Eliminate $\rightarrow$: $\neg(\neg A \vee \neg B)$
- Push negation using De Morgan: $(\neg\neg A \wedge \neg\neg B)$
- Use Double negation rule: $(A \wedge B)$

Disjunctive Normal Form (DNF)

A formula is said to be in Disjunctive Normal Form (DNF) if it is a disjunction of terms.

In other words, it is a formula of the form

$$\bigvee_i (\bigwedge_j l_{i,j})$$

where $l_{i,j}$ is the j-th literal in the i-th term.

Examples

- $(A \wedge \neg B \wedge C) \vee (\wedge A \wedge D) \vee (B)$ is in DNF.

Disjunctive Normal Form (DNF)

A formula is said to be in Disjunctive Normal Form (DNF) if it is a disjunction of terms.

In other words, it is a formula of the form

$$\bigvee_i \left(\bigwedge_j l_{i,j} \right)$$

where $l_{i,j}$ is the j-th literal in the i-th term.

Examples

- $(A \wedge \neg B \wedge C) \vee (\wedge A \wedge D) \vee (B)$ is in DNF.

DNF is a special case of NNF.

Coverting to DNF

Every formula can be converted to DNF in exponential time and space:

- Convert to NNF
- Distribute disjunctions following the rule:

$$\models A \wedge (B \vee C) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$$

Example: $(A \vee B) \wedge (\neg C \vee D)$

- $((A \vee B) \wedge (\neg C)) \vee ((A \vee B) \wedge D)$
- $(A \wedge \neg C) \vee (B \wedge \neg C) \vee (A \wedge D) \vee (B \wedge D)$

Coverting to DNF

Every formula can be converted to DNF in exponential time and space:

- Convert to NNF
- Distribute disjunctions following the rule:

$$\models A \wedge (B \vee C) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$$

Example: $(A \vee B) \wedge (\neg C \vee D)$

- $((A \vee B) \wedge (\neg C)) \vee ((A \vee B) \wedge D)$
- $(A \wedge \neg C) \vee (B \wedge \neg C) \vee (A \wedge D) \vee (B \wedge D)$

Q:How many clauses would the DNF have had we started from a conjunction of n clauses?

Satisfiability of DNF

Is the following DNF formula satisfiable?

$$(x_1 \wedge x_2 \wedge \neg x_1) \vee (x_2 \wedge x_1) \vee (x_2 \wedge \neg x_3 \wedge x_3)$$

Satisfiability of DNF

Is the following DNF formula satisfiable?

$$(x_1 \wedge x_2 \wedge \neg x_1) \vee (x_2 \wedge x_1) \vee (x_2 \wedge \neg x_3 \wedge x_3)$$

What is the complexity of satisfiability of DNF formulas?

Conjunctive Normal Form (CNF)

A formula is said to be in Conjunctive Normal Form (CNF) if it is a conjunction of clauses.

In other words, it is a formula of the form

$$\bigwedge_i \left(\bigvee_j l_{i,j} \right)$$

where $l_{i,j}$ is the j-th literal in the i-th term.

Examples

- $(A \vee \neg B \vee C) \wedge (\neg A \vee D) \wedge (B)$ is in CNF

Conjunctive Normal Form (CNF)

A formula is said to be in Conjunctive Normal Form (CNF) if it is a conjunction of clauses.

In other words, it is a formula of the form

$$\bigwedge_i \left(\bigvee_j l_{i,j} \right)$$

where $l_{i,j}$ is the j-th literal in the i-th term.

Examples

- $(A \vee \neg B \vee C) \wedge (\neg A \vee D) \wedge (B)$ is in CNF

CNF is a special case of NNF.

Coverting to CNF

Every formula can be converted to CNF:

Coverting to CNF

Every formula can be converted to CNF:

- in exponential time and space with the same set of atoms

Coverting to CNF

Every formula can be converted to CNF:

- in exponential time and space with the same set of atoms
- in linear time and space if new variables are added.

Coverting to CNF

Every formula can be converted to CNF:

- in exponential time and space with the same set of atoms
- in linear time and space if new variables are added.
  - In this case the original and converted formulas are "equi-satisfiable".
  - This technique is called Tseitin's encoding.

Converting to CNF: the Exponential Way

$\mathit{CNF}(\phi)\{$
case

- ϕ is a literal: return ϕ
- ϕ is $\varphi_1 \wedge \varphi_2$: return $\mathit{CNF}(\varphi_1) \wedge \mathit{CNF}(\varphi_2)$
- ϕ is $\varphi_1 \vee \varphi_2$: return $\mathit{Dist}(\mathit{CNF}(\varphi_1), \mathit{CNF}(\varphi_2))$

$\}$

$\mathit{Dist}(\varphi_1, \varphi_2)\{$
case

- φ_1 is $\psi_{11} \wedge \psi_{12}$: return $\mathit{Dist}(\psi_{11}, \varphi_2) \wedge \mathit{Dist}(\psi_{12}, \varphi_2)$
- φ_2 is $\psi_{21} \wedge \psi_{22}$: return $\mathit{Dist}(\varphi_1, \psi_{21}) \wedge \mathit{Dist}(\varphi_1, \psi_{22})$

$\}$

Converting to CNF: the Exponential Way

Consider the formula $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

Converting to CNF: the Exponential Way

Consider the formula $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

$$CNF(\phi) = (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (y_1 \vee x_2) \wedge (y_1 \vee y_2)$$

Converting to CNF: the Exponential Way

Consider the formula $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

$$CNF(\phi) = (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (y_1 \vee x_2) \wedge (y_1 \vee y_2)$$

Now consider: $\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$

Converting to CNF: the Exponential Way

Consider the formula $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

$$CNF(\phi) = (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (y_1 \vee x_2) \wedge (y_1 \vee y_2)$$

Now consider: $\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$

Q: How many clauses $CNF(\phi_n)$ returns?

Converting to CNF: the Exponential Way

Consider the formula $\phi = (x_1 \wedge y_1) \vee (x_2 \wedge y_2)$

$$CNF(\phi) = (x_1 \vee x_2) \wedge (x_1 \vee y_2) \wedge (y_1 \vee x_2) \wedge (y_1 \vee y_2)$$

Now consider: $\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$

Q: How many clauses $CNF(\phi_n)$ returns?

A: 2^n

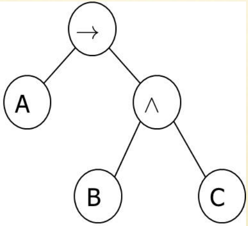
Tseitin's Encoding

Consider the formula $(A \rightarrow (B \wedge C))$

Tseitin's Encoding

Consider the formula $(A \rightarrow (B \wedge C))$

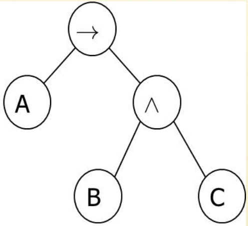
The parse tree:



Tseitin's Encoding

Consider the formula $(A \to (B \wedge C))$

The parse tree:



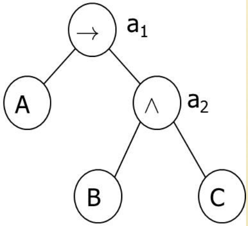
Associate a new auxiliary variable with each gate.

Add constraints that define these new variables.

Finally, enforce the root node.

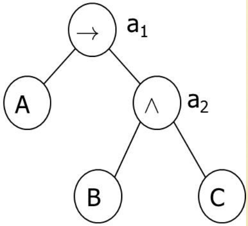
Tseitin's Encoding

$(a_1 \leftrightarrow (A \rightarrow a_2)) \wedge (a_2 \leftrightarrow$
$(B \wedge C)) \wedge (a_1)$



Tseitin's Encoding

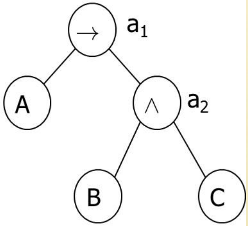
$(a_1 \leftrightarrow (A \rightarrow a_2)) \land (a_2 \leftrightarrow (B \land C)) \land (a_1)$



Each such constraint has a CNF representation with 3 or 4 clauses.

Tseitin's Encoding

$(a_1 \leftrightarrow (A \rightarrow a_2)) \wedge (a_2 \leftrightarrow (B \wedge C)) \wedge (a_1)$



Each such constraint has a CNF representation with 3 or 4 clauses.

First: $(a_1 \vee A) \wedge (a_1 \vee \neg a_2) \wedge (\neg a_1 \vee A \vee a_2)$

Second: $(\neg a_2 \vee B) \wedge (\neg a_2 \vee C) \wedge (a_2 \vee \neg B \vee \neg C)$

Tseitin's Encoding

$$\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

Tseitin's Encoding

$$\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

With Tseitin's encoding we need:

- n auxiliary variables $a_1, \ldots, a_n$.
- Each adds 3 constraints.
- Top clause: $(a_1 \vee \ldots \vee a_n)$

Tseitin's Encoding

$$\phi_n = (x_1 \wedge y_1) \vee (x_2 \wedge y_2) \vee \ldots \vee (x_n \wedge y_n)$$

With Tseitin's encoding we need:

- n auxiliary variables $a_1, \ldots, a_n$.
- Each adds 3 constraints.
- Top clause: $(a_1 \vee \ldots \vee a_n)$

Hence, we have

- $3n + 1$ clauses, instead of 2^n.
- $3n$ variables rather than $2n$.

SAT Problem and SAT Solver

SAT problem is: Given a Boolean formula in CNF, asking whether there exists an assignment to each variable so that the value of the formula is `true`.

SAT Problem and SAT Solver

SAT problem is: Given a Boolean formula in CNF, asking whether there exists an assignment to each variable so that the value of the formula is `true`.

It is a NPC problem, which means that there is only exponential algorithm so far. A SAT solver is a tool that solves the SAT problem.

SAT Problem and SAT Solver

SAT problem is: Given a Boolean formula in CNF, asking whether there exists an assignment to each variable so that the value of the formula is `true`.

It is a NPC problem, which means that there is only exponential algorithm so far. A SAT solver is a tool that solves the SAT problem. However,

SAT solver is to be said as the ”most successful formal tools, which can handle 100,000 variables with millions of clauses in less than one sec.