

Lecture_00_Motivation	2
Lecture_01_Natural_Deduction_01	48
Lecture_02_Natural_Deduction_02	118
Lecture_03_Soundness_and_Completeness_Of_PL	209
Lecture_04_Normal_Forms_and_Horn_Clauses	247
Lecture_05_Resolution_01	293
Lecture_06_Resolution_02	323
Lecture_07_Recap	363
Lecture_08_CNF_to_DNF	372
Lecture_09_DPLL	387
Lecture_10_FOL_Syntax	425
Lecture_11_FOL_Semantics	453
Lecture_12_FOL_SAT	491
Lecture_13_FOL_SAT_Translation	511
Lecture_14_FOL_RPF_Skolemisation	527
Lecture_15_FOL_Words	545
Lecture_16_FOL_FSM_01	581
Lecture_17_FOL_FSM_02	610

# **CS 228 : Logic in Computer Science**

Krishna. S

## Some Real Life Stories

# Therac-25(1987)

---



- ▶ The Therac-25 : radiation therapy machine produced by Atomic Energy of Canada Limited (AECL)

# Therac-25(1987)

---



- ▶ The Therac-25 : radiation therapy machine produced by Atomic Energy of Canada Limited (AECL)
- ▶ Involved in **at least six accidents**, in which patients were given massive overdoses of radiation, **approximately 100 times** the intended dose.

# Therac-25(1987)

---

- ▶ The Therac-25 : radiation therapy machine produced by Atomic Energy of Canada Limited (AECL)
- ▶ Involved in **at least six accidents**, in which patients were given massive overdoses of radiation, **approximately 100 times** the intended dose.
- ▶ Design error in the control software (race condition)

# Intel Pentium Bug (1994)

---



- ▶ The Intel FDIV bug : Bug in the intel P5 floating point unit

# Intel Pentium Bug (1994)

---



- ▶ The Intel FDIV bug : Bug in the intel P5 floating point unit
- ▶ Discovered by a professor working on Brun's constant
- ▶  $(\frac{1}{3} + \frac{1}{5}) + (\frac{1}{5} + \frac{1}{7}) + (\frac{1}{11} + \frac{1}{13}) + (\frac{1}{17} + \frac{1}{19}) + \dots$  converges to  $B \cong 1.90216054$

# Intel Pentium Bug (1994)

---



- ▶ The Intel FDIV bug : Bug in the intel P5 floating point unit
- ▶ Discovered by a professor working on Brun's constant
- ▶  $(\frac{1}{3} + \frac{1}{5}) + (\frac{1}{5} + \frac{1}{7}) + (\frac{1}{11} + \frac{1}{13}) + (\frac{1}{17} + \frac{1}{19}) + \dots$  converges to  $B \cong 1.90216054$
- ▶ Intel offered to replace all flawed processors

# Ariane 5 (1996)

---



- ▶ ESA (European Space Agency) **Ariane 5 Launcher**

# Ariane 5 (1996)

---



- ▶ ESA (European Space Agency) **Ariane 5 Launcher**
  - ▶ Shown here in maiden flight on 4th June 1996

# Ariane 5 (1996)

---



- ▶ ESA (European Space Agency) **Ariane 5 Launcher**
  - ▶ Shown here in maiden flight on 4th June 1996
- ▶ Self destructs 37 secs later

# Ariane 5 (1996)

---



- ▶ ESA (European Space Agency) **Ariane 5 Launcher**
  - ▶ Shown here in maiden flight on 4th June 1996
- ▶ Self destructs 37 secs later
  - ▶ uncaught exception: data conversion from 64-bit float to 16-bit signed int

# Toyota Prius (2010)

---



- ▶ First mass produced hybrid vehicle

# Toyota Prius (2010)

---



- ▶ First mass produced hybrid vehicle
  - ▶ software “glitch” found in anti-lock braking system
  - ▶ Eventually fixed via software update in total 185,000 cars recalled, at huge cost

# Nest Thermostat (2016)

---



- ▶ Nest Thermostat, the smart, learning thermostat from Nest Labs

# Nest Thermostat (2016)

---



- ▶ Nest Thermostat, the smart, learning thermostat from Nest Labs
  - ▶ software “glitch” led several homes to a frozen state, reported in NY times, Jan 13, 2016. May be, old fashioned mechanical thermostats better!

# What do these stories have in common?

---

- ▶ Programmable computing devices
  - ▶ conventional computers and networks
  - ▶ software embedded in devices

# What do these stories have in common?

---

- ▶ Programmable computing devices
  - ▶ conventional computers and networks
  - ▶ software embedded in devices
- ▶ Programming error direct cause of failure
- ▶ Software critical
  - ▶ for safety
  - ▶ for business
  - ▶ for performance

# What do these stories have in common?

---

- ▶ Programmable computing devices
  - ▶ conventional computers and networks
  - ▶ software embedded in devices
- ▶ Programming error direct cause of failure
- ▶ Software critical
  - ▶ for safety
  - ▶ for business
  - ▶ for performance
- ▶ High costs incurred: financial, loss of life
- ▶ Failures avoidable

# Formal Methods

---

## Intuitive Description

“Applied Mathematics for modelling and analysing ICT systems”

Formal methods offer a large potential for:

# Formal Methods

---

## Intuitive Description

“Applied Mathematics for modelling and analysing ICT systems”

Formal methods offer a large potential for:

- ▶ obtaining an **early integration** of verification in the design process

# Formal Methods

---

## Intuitive Description

“Applied Mathematics for modelling and analysing ICT systems”

Formal methods offer a large potential for:

- ▶ obtaining an **early integration** of verification in the design process
- ▶ providing **more effective** verification techniques (higher coverage)

# Formal Methods

---

## Intuitive Description

“Applied Mathematics for modelling and analysing ICT systems”

Formal methods offer a large potential for:

- ▶ obtaining an **early integration** of verification in the design process
- ▶ providing **more effective** verification techniques (higher coverage)
- ▶ **reducing** the verification time

# Simulation and Testing

---

## Basic procedure

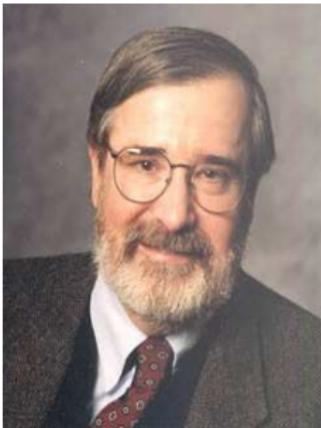
- ▶ Take a model
- ▶ Simulate it with certain inputs
- ▶ Observe what happens, and if this is desired

## Important Drawbacks

- ▶ possible behaviours very large/infinite
- ▶ unexplored behaviours may contain fatal bug
- ▶ can show presence of errors, **not** their absence

# Model Checking

---



- ▶ Year 2008 : ACM confers the **Turing Award** to the pioneers of Model Checking: **Ed Clarke, Allen Emerson, and Joseph Sifakis**
- ▶ Why?

# Model checking

---

- ▶ Model checking has evolved in last 25 years into a widely used verification and debugging technique for software and hardware.
- ▶ Cost of *not* doing formal verification is high!
  - ▶ The France Telecom example
  - ▶ Ariane rocket: kaboom due to integer overflow!
  - ▶ Toyota/Ford recalls

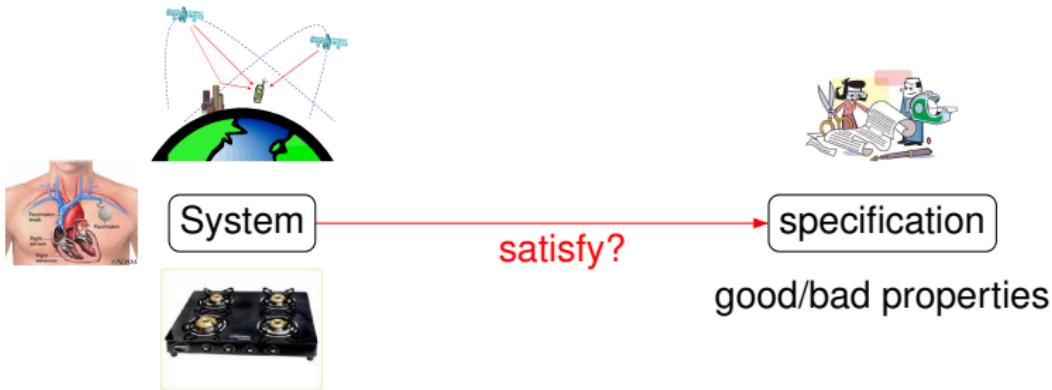
# Model checking

---

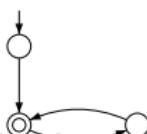
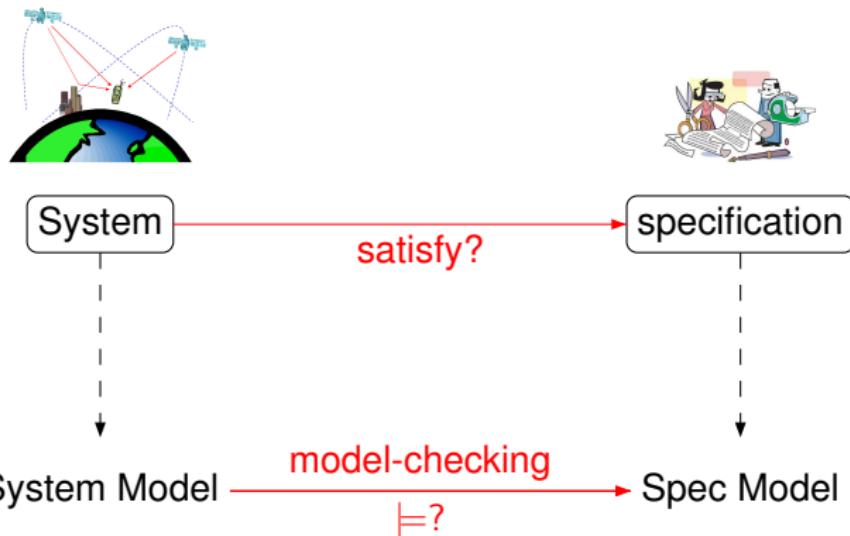
- ▶ Model checking has evolved in last 25 years into a widely used verification and debugging technique for software and hardware.
- ▶ Cost of *not* doing formal verification is high!
  - ▶ The France Telecom example
  - ▶ Ariane rocket: kaboom due to integer overflow!
  - ▶ Toyota/Ford recalls
- ▶ Model checking used (and further developed) by companies/institutes such as **IBM, Intel, NASA, Cadence, Microsoft, and Siemens**, and has culminated in many freely downloadable software tools that allow automated verification.

# What is Model Checking?

---



# What is Model Checking?



Lecture\_00\_Motivation

$\phi$

# Model Checker as a Black Box

---

- ▶ Inputs to Model checker : A finite state system  $M$ , and a property  $P$  to be checked.
- ▶ Question : Does  $M$  satisfy  $P$ ?
- ▶ Possible Outputs
  - ▶ Yes,  $M$  satisfies  $P$
  - ▶ No, here is a counter example!

# What are Models?

---

## Transition Systems

- ▶ States labeled with propositions
- ▶ Transition relation between states
- ▶ Action-labeled transitions to facilitate composition

# What are Models?

---

## Transition Systems

- ▶ States labeled with propositions
- ▶ Transition relation between states
- ▶ Action-labeled transitions to facilitate composition

## Expressivity

- ▶ Programs are transition systems
- ▶ Multi-threading programs are transition systems
- ▶ Communicating processes are transition systems
- ▶ Hardware circuits are transition systems
- ▶ What else?

# What are Properties?

---

## Example properties

- ▶ Can the system reach a deadlock?
- ▶ Can two processes ever be together in a critical section?
- ▶ On termination, does a program provide correct output?

# What are Properties?

## Example properties

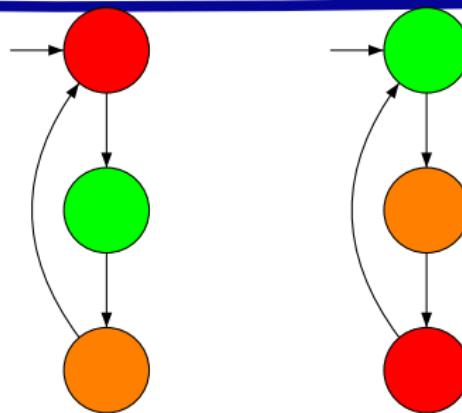
- ▶ Can the system reach a deadlock?
- ▶ Can two processes ever be together in a critical section?
- ▶ On termination, does a program provide correct output?

## Logics of Relevance

- ▶ Classical Logics
  - ▶ First Order Logic
  - ▶ Monadic Second Order Logic
- ▶ Temporal Logics
  - ▶ Propositional Logic, enriched with modal operators such as  $\Box$  (always) and  $\Diamond$  (eventually)
  - ▶ Interpreted over state sequences (linear)
  - ▶  $\Box$  over infinite trees (branching)

# Two Traffic Lights

---



1. The traffic lights are never green simultaneously  
 $\forall x(\neg(green_1(x) \wedge green_2(x)))$  or  $\square(\neg(green_1 \wedge green_2))$
2. The first traffic light is infinitely often green  
 $\forall x \exists y(x < y \wedge green_1(y))$  or  $\square \Diamond green_1$
3. Between every two occurrences of traffic light 1 becoming red, traffic light 2 becomes red once.

# The Model Checking Process

---

- ▶ Modeling Phase

- ▶ model the system under consideration
- ▶ as a first sanity check, perform some simulations
- ▶ formalise property to be checked

# The Model Checking Process

---

- ▶ Modeling Phase
  - ▶ model the system under consideration
  - ▶ as a first sanity check, perform some simulations
  - ▶ formalise property to be checked
- ▶ Running Phase
  - ▶ run the model checker to check the validity of the property in the model

# The Model Checking Process

---

- ▶ **Modeling Phase**
  - ▶ model the system under consideration
  - ▶ as a first sanity check, perform some simulations
  - ▶ formalise property to be checked
- ▶ **Running Phase**
  - ▶ run the model checker to check the validity of the property in the model
- ▶ **Analysis Phase**
  - ▶ property satisfied? → check next property (if any)
  - ▶ property violated?
    - ▶ analyse generated counter example by simulation
    - ▶ refine the model, design, property, ... and repeat entire procedure
  - ▶ out of memory? → try to reduce the model and try again

# The Pros of Model Checking

---

- ▶ widely applicable (hardware, software...)
- ▶ allows for partial verification (only relevant properties)
- ▶ potential “push-button” technology (tools)
- ▶ rapidly increasing industrial interest
- ▶ in case of property violation, a counter example is provided
- ▶ sound mathematical foundations
- ▶ not biased to the most possible scenarios (like testing)

# The Cons of Model Checking

---

- ▶ model checking is only as “good” as the system model
- ▶ no guarantee about **completeness** of results (incomplete specifications)

Nevertheless:

Model Checking is an effective technique to expose potential design errors

# Striking Model-Checking Examples

---

- ▶ Security : Needham-Schroeder encryption protocol
  - ▶ error that remained undiscovered for 17 years revealed (model checker SAL)
- ▶ Transportation Systems
  - ▶ Train model containing  $10^{47}$  states (model checker UPPAAL)
- ▶ Model Checkers for C, JAVA, C++
  - ▶ used (and developed) by Microsoft, Intel, NASA
  - ▶ successful application area: device drivers (model checker SLAM)
- ▶ Dutch storm surge barrier in Nieuwe Waterweg
- ▶ Software in current/next generation of space missiles
  - ▶ NASA's
    - ▶ Java Pathfinder, Deep Space Habitat, Lab for Reliable Software

# Relevant Topics

---

- ▶ What are appropriate **models**?
  - ▶ from programs, circuits, communication protocols to transition systems

# Relevant Topics

---

- ▶ What are appropriate **models**?
  - ▶ from programs, circuits, communication protocols to transition systems
- ▶ What are properties?
  - ▶ Safety, Liveness, fairness

# Relevant Topics

---

- ▶ What are appropriate **models**?
  - ▶ from programs, circuits, communication protocols to transition systems
- ▶ What are properties?
  - ▶ Safety, Liveness, fairness
- ▶ How to check **regular** properties?
  - ▶ finite state automata and regular safety properties
  - ▶ Büchi automata and  $\omega$ -regular properties

# Relevant Topics

---

- ▶ How to express properties **succinctly**?
  - ▶ First Order Logic (FO) : syntax, semantics
  - ▶ Monadic Second Order Logic (MSO) : syntax, semantics
  - ▶ Linear-Temporal-Logic (LTL) : syntax, semantics
  - ▶ What can be expressed in each logic?
  - ▶ Satisfiability and Model checking : algorithms, complexity

# Relevant Topics

---

- ▶ How to express properties **succinctly**?
  - ▶ First Order Logic (FO) : syntax, semantics
  - ▶ Monadic Second Order Logic (MSO) : syntax, semantics
  - ▶ Linear-Temporal-Logic (LTL) : syntax, semantics
  - ▶ What can be expressed in each logic?
  - ▶ Satisfiability and Model checking : algorithms, complexity
- ▶ How to make models **succint**?
  - ▶ Equivalences and partial-orders on transition systems
  - ▶ Which properties are preserved?
  - ▶ Minimization algorithms

# CS 228 : Logic in Computer Science

Krishna. S

# Welcome

---

What is this course about? A mini-zoo of logics.

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?
- ▶ Q3: How easy is to answer Q1 and Q2?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?
- ▶ Q3: How easy is to answer Q1 and Q2?
- ▶ Q4: Can you write an algorithm to answer Q1 and Q2?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?
- ▶ Q3: How easy is to answer Q1 and Q2?
- ▶ Q4: Can you write an algorithm to answer Q1 and Q2?
- ▶ Q5: Can you “prove” any factually correct statement using the chosen logic  $L$ ?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?
- ▶ Q3: How easy is to answer Q1 and Q2?
- ▶ Q4: Can you write an algorithm to answer Q1 and Q2?
- ▶ Q5: Can you “prove” any factually correct statement using the chosen logic  $L$ ?
- ▶ Q6: How is logic  $L$  used in computer science?

# Welcome

---

What is this course about? A mini-zoo of logics. Here are some typical questions you will learn to answer:

- ▶ Q1: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  satisfiable?
- ▶ Q2: Given a formula  $\varphi$  in a logic  $L$ , is  $\varphi$  valid?
- ▶ Q3: How easy is to answer Q1 and Q2?
- ▶ Q4: Can you write an algorithm to answer Q1 and Q2?
- ▶ Q5: Can you “prove” any factually correct statement using the chosen logic  $L$ ?
- ▶ Q6: How is logic  $L$  used in computer science?
- ▶ Q7: What are the techniques needed to go about these questions?

# Some Members of the mini-zoo

---

- ▶ Propositional Logic
- ▶ First Order Logic
- ▶ Monadic Second Order Logic
- ▶ Propositional Dynamic Logic
- ▶ Linear Temporal Logic
- ▶ Computational Tree Logic

More if time permits!

# References

---

- ▶ To start with, the text book of Huth and Ryan : Logic for CS.
- ▶ As we go ahead, lecture notes/monographs/other text books.
- ▶ Classes : Slot 4. Tutorial: To discuss.

# Propositional Logic

# Syntax

---

- ▶ Finite set of propositional variables  $p, q, \dots$

# Syntax

---

- ▶ Finite set of propositional variables  $p, q, \dots$
- ▶ Each of these can be true/false

# Syntax

---

- ▶ Finite set of propositional variables  $p, q, \dots$
- ▶ Each of these can be true/false
- ▶ Combine propositions using  $\neg, \vee, \wedge, \rightarrow$

# Syntax

---

- ▶ Finite set of propositional variables  $p, q, \dots$
- ▶ Each of these can be true/false
- ▶ Combine propositions using  $\neg, \vee, \wedge, \rightarrow$
- ▶ Parentheses as required

# Syntax

---

- ▶ Finite set of propositional variables  $p, q, \dots$
- ▶ Each of these can be true/false
- ▶ Combine propositions using  $\neg, \vee, \wedge, \rightarrow$
- ▶ Parentheses as required
- ▶ Example :  $[p \wedge (q \vee r)] \rightarrow [\neg r \wedge p]$
- ▶  $\neg$  binds tighter than  $\vee, \wedge$ , which bind tighter than  $\rightarrow$ . In the absence of parentheses,  $p \rightarrow q \rightarrow r$  is read as  $p \rightarrow (q \rightarrow r)$

# Natural Deduction

---

- ▶ If it rains, Alice is outside and does not have any raingear with her, she will get wet.  $\varphi = (R \wedge \text{AliceOut} \wedge \neg RG) \rightarrow \text{AliceWet}$

# Natural Deduction

---

- ▶ If it rains, Alice is outside and does not have any raingear with her, she will get wet.  $\varphi = (R \wedge \text{AliceOut} \wedge \neg RG) \rightarrow \text{AliceWet}$
- ▶ It is raining, and Alice is outside, and is not wet.  
 $\psi = (R \wedge \text{AliceOut} \wedge \neg \text{AliceWet})$

# Natural Deduction

---

- ▶ If it rains, Alice is outside and does not have any raingear with her, she will get wet.  $\varphi = (R \wedge \text{AliceOut} \wedge \neg RG) \rightarrow \text{AliceWet}$
- ▶ It is raining, and Alice is outside, and is not wet.  
 $\psi = (R \wedge \text{AliceOut} \wedge \neg \text{AliceWet})$
- ▶ So, Alice has her rain gear with her.  $RG$
- ▶ Thus,  $\chi = \varphi \wedge \psi \rightarrow RG$ . You can deduce  $RG$  from  $\varphi \wedge \psi$ .
- ▶ Is  $\chi$  valid? Is  $\chi$  satisfiable?

## Two Examples of Natural Deduction

# Solve Sudoku

---

Consider the following kid's version of Sudoku.

	2	4	
1			3
4			2
	1	3	

Rules:

- ▶ Each row must contain all numbers 1-4
- ▶ Each column must contain all numbers 1-4
- ▶ Each  $2 \times 2$  block must contain all numbers 1-4
- ▶ No cell contains 2 or more numbers

# Encoding as Propositional Satisfiability

---

- ▶ Proposition  $P(i, j, n)$  is true when cell  $(i, j)$  has number  $n$

# Encoding as Propositional Satisfiability

---

- ▶ Proposition  $P(i, j, n)$  is true when cell  $(i, j)$  has number  $n$
- ▶  $4 \times 4 \times 4$  propositions

# Encoding as Propositional Satisfiability

---

- ▶ Proposition  $P(i, j, n)$  is true when cell  $(i, j)$  has number  $n$
- ▶  $4 \times 4 \times 4$  propositions
- ▶ **Each row must contain all 4 numbers**
  - ▶ Row 1:  $[P(1, 1, 1) \vee P(1, 2, 1) \vee P(1, 3, 1) \vee P(1, 4, 1)] \wedge [P(1, 1, 2) \vee P(1, 2, 2) \vee P(1, 3, 2) \vee P(1, 4, 2)] \wedge [P(1, 1, 3) \vee P(1, 2, 3) \vee P(1, 3, 3) \vee P(1, 4, 3)] \wedge [P(1, 1, 4) \vee P(1, 2, 4) \vee P(1, 3, 4) \vee P(1, 4, 4)]$

# Encoding as Propositional Satisfiability

---

- ▶ Proposition  $P(i, j, n)$  is true when cell  $(i, j)$  has number  $n$
- ▶  $4 \times 4 \times 4$  propositions
- ▶ **Each row must contain all 4 numbers**
  - ▶ Row 1:  $[P(1, 1, 1) \vee P(1, 2, 1) \vee P(1, 3, 1) \vee P(1, 4, 1)] \wedge [P(1, 1, 2) \vee P(1, 2, 2) \vee P(1, 3, 2) \vee P(1, 4, 2)] \wedge [P(1, 1, 3) \vee P(1, 2, 3) \vee P(1, 3, 3) \vee P(1, 4, 3)] \wedge [P(1, 1, 4) \vee P(1, 2, 4) \vee P(1, 3, 4) \vee P(1, 4, 4)]$
  - ▶ Row 2:  $[P(2, 1, 1) \vee \dots]$
  - ▶ Row 3:  $[P(3, 1, 1) \vee \dots]$
  - ▶ Row 4:  $[P(4, 1, 1) \vee \dots]$

# Encoding as Propositional Satisfiability

---

Each column must contain all numbers 1-4

# Encoding as Propositional Satisfiability

---

Each column must contain all numbers 1-4

- ▶ Column 1:  $[P(1, 1, 1) \vee P(2, 1, 1) \vee P(3, 1, 1) \vee P(4, 1, 1)] \wedge [P(1, 1, 2) \vee P(2, 1, 2) \vee P(3, 1, 2) \vee P(4, 1, 2)] \wedge [P(1, 1, 3) \vee P(2, 1, 3) \vee P(3, 1, 3) \vee P(4, 1, 3)] \wedge [P(1, 1, 4) \vee P(2, 1, 4) \vee P(3, 1, 4) \vee P(4, 1, 4)]$

# Encoding as Propositional Satisfiability

---

Each column must contain all numbers 1-4

- ▶ Column 1:  $[P(1, 1, 1) \vee P(2, 1, 1) \vee P(3, 1, 1) \vee P(4, 1, 1)] \wedge [P(1, 1, 2) \vee P(2, 1, 2) \vee P(3, 1, 2) \vee P(4, 1, 2)] \wedge [P(1, 1, 3) \vee P(2, 1, 3) \vee P(3, 1, 3) \vee P(4, 1, 3)] \wedge [P(1, 1, 4) \vee P(2, 1, 4) \vee P(3, 1, 4) \vee P(4, 1, 4)]$
- ▶ Column 2:  $[P(1, 2, 1) \vee \dots]$
- ▶ Column 3:  $[P(1, 3, 1) \vee \dots]$
- ▶ Column 4:  $[P(1, 4, 1) \vee \dots]$

# Encoding as Propositional Satisfiability

---

Each  $2 \times 2$  block must contain all numbers 1-4

# Encoding as Propositional Satisfiability

---

Each  $2 \times 2$  block must contain all numbers 1-4

- ▶ Upper left block contains all numbers 1-4:

$$\begin{aligned}[P(1, 1, 1) \vee P(1, 2, 1) \vee P(2, 1, 1) \vee P(2, 2, 1)] \wedge \\ [P(1, 1, 2) \vee P(1, 2, 2) \vee P(2, 1, 2) \vee P(2, 2, 2)] \wedge \\ [P(1, 1, 3) \vee P(1, 2, 3) \vee P(2, 1, 3) \vee P(2, 2, 3)] \wedge \\ [P(1, 1, 4) \vee P(1, 2, 4) \vee P(2, 1, 4) \vee P(2, 2, 4)]\end{aligned}$$

# Encoding as Propositional Satisfiability

---

Each  $2 \times 2$  block must contain all numbers 1-4

- ▶ Upper left block contains all numbers 1-4:

$$\begin{aligned}[P(1, 1, 1) \vee P(1, 2, 1) \vee P(2, 1, 1) \vee P(2, 2, 1)] \wedge \\ [P(1, 1, 2) \vee P(1, 2, 2) \vee P(2, 1, 2) \vee P(2, 2, 2)] \wedge \\ [P(1, 1, 3) \vee P(1, 2, 3) \vee P(2, 1, 3) \vee P(2, 2, 3)] \wedge \\ [P(1, 1, 4) \vee P(1, 2, 4) \vee P(2, 1, 4) \vee P(2, 2, 4)]\end{aligned}$$

- ▶ Upper right block contains all numbers 1-4:

$$[P(1, 3, 1) \vee P(1, 4, 1) \vee P(2, 3, 1) \vee P(2, 4, 1)] \wedge \dots$$

- ▶ Lower left block contains all numbers 1-4:

$$[P(3, 1, 1) \vee P(3, 2, 1) \vee P(4, 1, 1) \vee P(4, 2, 1)] \wedge \dots$$

- ▶ Lower right block contains all numbers 1-4:

$$\text{Lecture\_01\_Natural\_Deduction\_01} \quad [P(3, 3, 1) \vee P(3, 4, 1) \vee P(4, 3, 1) \vee P(4, 4, 1)] \wedge \dots$$

# Encoding as Propositional Satisfiability

---

No cell contains 2 or more numbers

- ▶ For cell(1,1):

$$P(1, 1, 1) \rightarrow [\neg P(1, 1, 2) \wedge \neg P(1, 1, 3) \wedge \neg P(1, 1, 4)] \wedge$$

$$P(1, 1, 2) \rightarrow [\neg P(1, 1, 1) \wedge \neg P(1, 1, 3) \wedge \neg P(1, 1, 4)] \wedge$$

$$P(1, 1, 3) \rightarrow [\neg P(1, 1, 1) \wedge \neg P(1, 1, 2) \wedge \neg P(1, 1, 4)] \wedge$$

$$P(1, 1, 4) \rightarrow [\neg P(1, 1, 1) \wedge \neg P(1, 1, 2) \wedge \neg P(1, 1, 3)] \wedge$$

- ▶ Similar for other cells

# Encoding as Propositional Satisfiability

---

Encoding Initial Configuration:

$$P(1, 2, 2) \wedge P(1, 3, 4) \wedge P(2, 1, 1) \wedge P(2, 4, 3) \wedge \\ P(3, 1, 4) \wedge P(3, 4, 2) \wedge P(4, 2, 1) \wedge P(4, 3, 3)$$

## Solving Sudoku

To solve the puzzle, just conjunct all the above formulae and find a satisfiable truth assignment!

# Gold Rush

---

- (Box1) *The gold is not here*
- (Box2) *The gold is not here*
- (Box3) *The gold is in Box 2*

Only one message is true; the other two are false. Which box has the gold?

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1$ ,

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2,$

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3),$

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3), M_1 \vee M_2 \vee M_3,$

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3), M_1 \vee M_2 \vee M_3,$
  - ▶  $(\neg M_1 \wedge \neg M_2) \vee (\neg M_1 \wedge \neg M_3) \vee (\neg M_2 \wedge \neg M_3)$

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3), M_1 \vee M_2 \vee M_3,$
  - ▶  $(\neg M_1 \wedge \neg M_2) \vee (\neg M_1 \wedge \neg M_3) \vee (\neg M_2 \wedge \neg M_3)$
  - ▶ Conjunct all these, and call the formula  $\varphi$ .

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3), M_1 \vee M_2 \vee M_3,$
  - ▶  $(\neg M_1 \wedge \neg M_2) \vee (\neg M_1 \wedge \neg M_3) \vee (\neg M_2 \wedge \neg M_3)$
  - ▶ Conjunct all these, and call the formula  $\varphi$ .
  - ▶ Is there a unique satisfiable assignment for  $\varphi$ ?

# Solve Gold Rush

---

- ▶ Propositions  $M_1, M_2, M_3$  representing messages in boxes 1,2,3
- ▶ Propositions  $G_1, G_2, G_3$  representing gold in boxes 1,2,3
- ▶ Formalize what is given to you
  - ▶  $M_1 \leftrightarrow \neg G_1, M_2 \leftrightarrow \neg G_2, M_3 \leftrightarrow G_2$
  - ▶  $\neg(M_1 \wedge M_2 \wedge M_3), M_1 \vee M_2 \vee M_3,$
  - ▶  $(\neg M_1 \wedge \neg M_2) \vee (\neg M_1 \wedge \neg M_3) \vee (\neg M_2 \wedge \neg M_3)$
  - ▶ Conjunct all these, and call the formula  $\varphi$ .
  - ▶ Is there a unique satisfiable assignment for  $\varphi$ ?
  - ▶ For example, is  $M_1 = \text{true}$  a part of the satisfiable assignment?

# A Proof Engine for Natural Deduction

---

- ▶ If it rains, Alice is outside and does not have any raingear with her, she will get wet.  $\varphi = (R \wedge \text{AliceOut} \wedge \neg RG) \rightarrow \text{AliceWet}$
- ▶ It is raining, and Alice is outside, and is not wet.  
 $\psi = (R \wedge \text{AliceOut} \wedge \neg \text{AliceWet})$
- ▶ So, Alice has her rain gear with her.  $RG$
- ▶ Thus,  $\chi = \varphi \wedge \psi \rightarrow RG$ .
- ▶ Given  $\varphi, \psi$ , can we “prove”  $RG$ ?

# A Proof Engine

---

- ▶ Given a formula  $\varphi$  in propositional logic, how to “prove”  $\varphi$  if  $\varphi$  is valid?
- ▶ What is a proof engine?
- ▶ Show that this proof engine is sound and complete
  - ▶ Completeness: Any fact that can be captured using propositional logic can be proved by the proof engine
  - ▶ Soundness: Any formula that is proved to be valid by the proof engine is indeed valid

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**
- ▶ These proof rules allow us to infer formulae from some given formulae

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**
- ▶ These proof rules allow us to infer formulae from some given formulae
- ▶ Given a set of **premises**, we **deduce** a **conclusion** which is also a formula using proof rules.

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**
- ▶ These proof rules allow us to infer formulae from some given formulae
- ▶ Given a set of **premises**, we **deduce** a **conclusion** which is also a formula using proof rules.
- ▶  $\varphi_1, \dots, \varphi_n \vdash \psi$  : This is called a **sequent**.  $\varphi_1, \dots, \varphi_n$  are **premises**, and  $\psi$ , the **conclusion**.
- ▶ Given  $\varphi_1, \dots, \varphi_n$ , we can deduce or prove  $\psi$ . **What was the sequent in the Alice example?**

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**
- ▶ These proof rules allow us to infer formulae from some given formulae
- ▶ Given a set of **premises**, we **deduce** a **conclusion** which is also a formula using proof rules.
- ▶  $\varphi_1, \dots, \varphi_n \vdash \psi$  : This is called a **sequent**.  $\varphi_1, \dots, \varphi_n$  are **premises**, and  $\psi$ , the **conclusion**.
- ▶ Given  $\varphi_1, \dots, \varphi_n$ , we can deduce or prove  $\psi$ . **What was the sequent in the Alice example?**
- ▶ For example,  $\neg p \rightarrow q, q \rightarrow r, \neg r \vdash p$  is a sequent. How do you prove this?

# Natural Deduction

---

- ▶ In natural deduction, we have a collection of **proof rules**
- ▶ These proof rules allow us to infer formulae from some given formulae
- ▶ Given a set of **premises**, we **deduce** a **conclusion** which is also a formula using proof rules.
- ▶  $\varphi_1, \dots, \varphi_n \vdash \psi$  : This is called a **sequent**.  $\varphi_1, \dots, \varphi_n$  are **premises**, and  $\psi$ , the **conclusion**.
- ▶ Given  $\varphi_1, \dots, \varphi_n$ , we can deduce or prove  $\psi$ . **What was the sequent in the Alice example?**
- ▶ For example,  $\neg p \rightarrow q, q \rightarrow r, \neg r \vdash p$  is a sequent. How do you prove this?
- ▶ Proof rules to be carefully chosen, for instance you should not end up proving something like  $p \wedge q \vdash \neg q$

## The Rules of the Proof Engine

# Rules for Natural Deduction

---

The **and introduction rule** denoted  $\wedge i$

$$\frac{\varphi \quad \psi}{\varphi \wedge \psi}$$

# Rules for Natural Deduction

---

The **and elimination rule** denoted  $\wedge e_1$

$$\frac{\varphi \wedge \psi}{\varphi}$$

The **and elimination rule** denoted  $\wedge e_2$

$$\frac{\varphi \wedge \psi}{\psi}$$

## A first proof using $\wedge i$ , $\wedge e_1$ , $\wedge e_2$

---

- ▶ Show that  $p \wedge q, r \vdash q \wedge r$ 
  1.  $p \wedge q$  premise
  - 2.

## A first proof using $\wedge i$ , $\wedge e_1$ , $\wedge e_2$

---

- ▶ Show that  $p \wedge q, r \vdash q \wedge r$

1.  $p \wedge q$  premise
2.  $r$  premise
- 3.

## A first proof using $\wedge i$ , $\wedge e_1$ , $\wedge e_2$

---

- ▶ Show that  $p \wedge q, r \vdash q \wedge r$

1.  $p \wedge q$  premise
2.  $r$  premise
3.  $q$   $\wedge e_2$  1
- 4.

## A first proof using $\wedge i$ , $\wedge e_1$ , $\wedge e_2$

---

- ▶ Show that  $p \wedge q, r \vdash q \wedge r$

1.  $p \wedge q$  premise
2.  $r$  premise
3.  $q$   $\wedge e_2$  1
4.  $q \wedge r$   $\wedge i$  3,2

# Rules for Natural Deduction

---

The rule of double negation elimination  $\neg\neg e$

$$\frac{\neg\neg\varphi}{\varphi}$$

The rule of double negation introduction  $\neg\neg i$

$$\frac{\varphi}{\neg\neg\varphi}$$

# Rules for Natural Deduction

---

The implies elimination rule or Modus Ponens MP

$$\frac{\varphi \quad \varphi \rightarrow \psi}{\psi}$$

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$ 
  1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
  - 2.

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$ 
  1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
  2.  $p \rightarrow q$  premise
  - 3.

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$

1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
2.  $p \rightarrow q$  premise
3.  $p$  premise
- 4.

# Another Proof

---

► Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$

1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
2.  $p \rightarrow q$  premise
3.  $p$  premise
4.  $q \rightarrow \neg\neg r$  MP 1,3
- 5.

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$

1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
2.  $p \rightarrow q$  premise
3.  $p$  premise
4.  $q \rightarrow \neg\neg r$  MP 1,3
5.  $q$  MP 2,3
- 6.

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$

1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
2.  $p \rightarrow q$  premise
3.  $p$  premise
4.  $q \rightarrow \neg\neg r$  MP 1,3
5.  $q$  MP 2,3
6.  $\neg\neg r$  MP 4,5
- 7.

# Another Proof

---

- ▶ Show that  $p, p \rightarrow q, p \rightarrow (q \rightarrow \neg\neg r) \vdash r$

1.  $p \rightarrow (q \rightarrow \neg\neg r)$  premise
2.  $p \rightarrow q$  premise
3.  $p$  premise
4.  $q \rightarrow \neg\neg r$  MP 1,3
5.  $q$  MP 2,3
6.  $\neg\neg r$  MP 4,5
7.  $r$   $\neg\neg e$  6

# CS 228 : Logic in Computer Science

Krishna. S

# Rules for Natural Deduction

---

Another implies elimination rule or Modus Tollens MT

$$\frac{\varphi \rightarrow \psi \quad \neg\psi}{\neg\varphi}$$

# A Proof

---

- ▶ Show that  $p \rightarrow \neg q, q \vdash \neg p$ 
  1.  $p \rightarrow \neg q$  premise
  - 2.

# A Proof

---

- ▶ Show that  $p \rightarrow \neg q, q \vdash \neg p$

1.  $p \rightarrow \neg q$  premise
2.  $q$  premise
- 3.

# A Proof

---

- ▶ Show that  $p \rightarrow \neg q, q \vdash \neg p$

1.  $p \rightarrow \neg q$  premise
2.  $q$  premise
3.  $\neg\neg q$   $\neg\neg i$  2
- 4.

# A Proof

---

- ▶ Show that  $p \rightarrow \neg q, q \vdash \neg p$

1.  $p \rightarrow \neg q$  premise
2.  $q$  premise
3.  $\neg\neg q$   $\neg\neg i$  2
4.  $\neg p$  MT 1,3

# More Rules

---

- ▶ Thanks to MT, we have  $p \rightarrow q, \neg q \vdash \neg p$ .

# More Rules

---

- ▶ Thanks to MT, we have  $p \rightarrow q, \neg q \vdash \neg p$ .
- ▶ Can we prove  $p \rightarrow q \vdash \neg q \rightarrow \neg p$ ?

# More Rules

---

- ▶ Thanks to MT, we have  $p \rightarrow q, \neg q \vdash \neg p$ .
- ▶ Can we prove  $p \rightarrow q \vdash \neg q \rightarrow \neg p$ ?
- ▶ So far, no proof rule that can do this.

# More Rules

---

- ▶ Thanks to MT, we have  $p \rightarrow q, \neg q \vdash \neg p$ .
- ▶ Can we prove  $p \rightarrow q \vdash \neg q \rightarrow \neg p$ ?
- ▶ So far, no proof rule that can do this.
- ▶ Given  $p \rightarrow q$ , let us assume  $\neg q$ . Can we then prove  $\neg p$ ?

# More Rules

---

- ▶ Thanks to MT, we have  $p \rightarrow q, \neg q \vdash \neg p$ .
- ▶ Can we prove  $p \rightarrow q \vdash \neg q \rightarrow \neg p$ ?
- ▶ So far, no proof rule that can do this.
- ▶ Given  $p \rightarrow q$ , let us assume  $\neg q$ . Can we then prove  $\neg p$ ?
- ▶ Yes, using MT.

# The implies introduction rule $\rightarrow i$

---

- ▶  $p \rightarrow q \vdash \neg q \rightarrow \neg p$

1.	$p \rightarrow q$	premise
2.	$\neg q$	assumption
3.	$\neg p$	MT 1,2
4.	$\neg q \rightarrow \neg p$	$\rightarrow i$ 2-3

## More on $\rightarrow$ i

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.      *true*
- 2.

premise

## More on $\rightarrow$ i

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.		

## More on $\rightarrow$ i

---

- ▶  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.		

## More on $\rightarrow$ i

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.		

## More on $\rightarrow$ i

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.		

# More on $\rightarrow i$

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.		

## More on $\rightarrow i$

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.	$q$	$\neg\neg e$ 6
8.		

## More on $\rightarrow i$

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.	$q$	$\neg\neg e$ 6
8.	$r$	MP 2,7

# More on $\rightarrow i$

---

►  $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.	$q$	$\neg\neg e$ 6
8.	$r$	MP 2,7
9.	$p \rightarrow r$	$\rightarrow i$ 4-8

# More on $\rightarrow$ i

- $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.	$q$	$\neg\neg e$ 6
8.	$r$	MP 2,7
9.	$p \rightarrow r$	$\rightarrow i$ 4-8
10.	$(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)$	$\rightarrow i$ 3-9
11.		

# More on $\rightarrow i$

- $\vdash (q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$

1.	<i>true</i>	premise
2.	$q \rightarrow r$	assumption
3.	$\neg q \rightarrow \neg p$	assumption
4.	$p$	assumption
5.	$\neg\neg p$	$\neg\neg i$ 4
6.	$\neg\neg q$	MT 3,5
7.	$q$	$\neg\neg e$ 6
8.	$r$	MP 2,7
9.	$p \rightarrow r$	$\rightarrow i$ 4-8
10.	$(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)$	$\rightarrow i$ 3-9
11.	$(q \rightarrow r) \rightarrow [(\neg q \rightarrow \neg p) \rightarrow (p \rightarrow r)]$	$\rightarrow i$ 2-10

# Transforming Proofs

---

- ▶  $(q \rightarrow r), (\neg q \rightarrow \neg p), p \vdash r$
- ▶ Transform any proof  $\varphi_1, \dots, \varphi_n \vdash \psi$  to  
 $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow \dots (\varphi_n \rightarrow \psi) \dots)$  by adding  $n$  lines of the rule  $\rightarrow i$

# More Examples

---

- ▶  $p \rightarrow (q \rightarrow r) \vdash (p \wedge q) \rightarrow r$ 
  1.  $p \rightarrow (q \rightarrow r)$  premise
  - 2.

# More Examples

---

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

1.  $p \rightarrow (q \rightarrow r)$  premise
2.  $p \wedge q$  assumption
- 3.

# More Examples

---

- ▶  $p \rightarrow (q \rightarrow r) \vdash (p \wedge q) \rightarrow r$

1.	$p \rightarrow (q \rightarrow r)$	premise
2.	$p \wedge q$	assumption
3.	$p$	$\wedge e_1 2$
4.		

# More Examples

---

- ▶  $p \rightarrow (q \rightarrow r) \vdash (p \wedge q) \rightarrow r$

1.	$p \rightarrow (q \rightarrow r)$	premise
2.	$p \wedge q$	assumption
3.	$p$	$\wedge e_1 2$
4.	$q$	$\wedge e_2 2$
5.		

# More Examples

---

- ▶  $p \rightarrow (q \rightarrow r) \vdash (p \wedge q) \rightarrow r$

1.	$p \rightarrow (q \rightarrow r)$	premise
2.	$p \wedge q$	assumption
3.	$p$	$\wedge e_1$ 2
4.	$q$	$\wedge e_2$ 2
5.	$q \rightarrow r$	MP 1,3
6.		

# More Examples

---

- ▶  $p \rightarrow (q \rightarrow r) \vdash (p \wedge q) \rightarrow r$

1.	$p \rightarrow (q \rightarrow r)$	premise
2.	$p \wedge q$	assumption
3.	$p$	$\wedge e_1$ 2
4.	$q$	$\wedge e_2$ 2
5.	$q \rightarrow r$	MP 1,3
6.	$r$	MP 4,5
7.		

# More Examples

---

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

1.	$p \rightarrow (q \rightarrow r)$	premise
2.	$p \wedge q$	assumption
3.	$p$	$\wedge e_1$ 2
4.	$q$	$\wedge e_2$ 2
5.	$q \rightarrow r$	MP 1,3
6.	$r$	MP 4,5
7.	$p \wedge q \rightarrow r$	$\rightarrow i$ 2-6

# More Rules

---

The or introduction rule  $\vee i_1$

$$\frac{\varphi}{\varphi \vee \psi}$$

The or introduction rule  $\vee i_2$

$$\frac{\psi}{\varphi \vee \psi}$$

# More Rules

---

The or elimination rule  $\vee e$

$$\frac{\varphi \vee \psi \quad \varphi \vdash \chi \quad \psi \vdash \chi}{\chi}$$

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.      $q \rightarrow r$                               premise

2.

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.  $q \rightarrow r$  premise
2.  $p \vee q$  assumption
- 3.

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.  $q \rightarrow r$  premise
2.  $p \vee q$  assumption
3.  $p$   $\vee\ e\ (1)$
- 4.

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e\ (1)$
4.	$p \vee r$	$\vee i_1\ 3$
5.		

# Or Elimination Example

---

- ▶  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e\ (1)$
4.	$p \vee r$	$\vee i_1\ 3$
5.	$q$	$\vee e\ (2)$
6.		

# Or Elimination Example

---

- ▶  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e\ (1)$
4.	$p \vee r$	$\vee i_1\ 3$
5.	$q$	$\vee e\ (2)$
6.	$r$	MP 1,5
7.		

# Or Elimination Example

---

- ▶  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e$ (1)
4.	$p \vee r$	$\vee i_1$ 3
5.	$q$	$\vee e$ (2)
6.	$r$	MP 1,5
7.	$p \vee r$	$\vee i_2$ 6

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e$ (1)
4.	$p \vee r$	$\vee i_1$ 3
5.	$q$	$\vee e$ (2)
6.	$r$	MP 1,5
7.	$p \vee r$	$\vee i_2$ 6
8.	$p \vee r$	$\vee e$ 2, 3-4, 5-7

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e$ (1)
4.	$p \vee r$	$\vee i_1$ 3
5.	$q$	$\vee e$ (2)
6.	$r$	MP 1,5
7.	$p \vee r$	$\vee i_2$ 6
8.	$p \vee r$	$\vee e$ 2, 3-4, 5-7
9.		

# Or Elimination Example

---

►  $q \rightarrow r \vdash (p \vee q) \rightarrow (p \vee r)$

1.	$q \rightarrow r$	premise
2.	$p \vee q$	assumption
3.	$p$	$\vee e$ (1)
4.	$p \vee r$	$\vee i_1$ 3
5.	$q$	$\vee e$ (2)
6.	$r$	MP 1,5
7.	$p \vee r$	$\vee i_2$ 6
8.	$p \vee r$	$\vee e$ 2, 3-4, 5-7
9.	$(p \vee q) \rightarrow (p \vee r)$	$\rightarrow i$ 2-8

# Associativity Using Or Elimination

---

- ▶  $(p \vee q) \vee r \vdash p \vee (q \vee r)$ 
  1.  $(p \vee q) \vee r$  premise
  - 2.

# Associativity Using Or Elimination

---

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise
2.  $\boxed{p \vee q \quad \vee e (1)}$
- 3.

# Associativity Using Or Elimination

---

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.

# Associativity Using Or Elimination

---

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e (1)$

3.  $p$   $\vee e (1.1)$

4.  $p \vee (q \vee r)$   $\vee i_1 3$

5.

# Associativity Using Or Elimination

---

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.  $q \vee r$   $\vee i_1$  5

7.

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.  $q \vee r$   $\vee i_1$  5

7.  $p \vee (q \vee r)$   $\vee i_2$  6

8.

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.  $q \vee r$   $\vee i_1$  5

7.  $p \vee (q \vee r)$   $\vee i_2$  6

8.  $p \vee (q \vee r)$   $\vee e$  2, 3-4, 5-7

9.

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise
2.  $p \vee q$   $\vee e$  (1)
3.  $p$   $\vee e$  (1.1)  
[ $p$ ]  
 $p \vee (q \vee r)$   $\vee i_1$  3
4.  $p \vee (q \vee r)$   $\vee i_1$  3
5.  $q$   $\vee e$  (1.2)  
[ $q$ ]  
 $q \vee r$   $\vee i_1$  5
6.  $q \vee r$   $\vee i_1$  5
7.  $p \vee (q \vee r)$   $\vee i_2$  6  
[ $p \vee (q \vee r)$ ]  
 $p \vee (q \vee r)$   $\vee e$  2, 3-4, 5-7
8.  $p \vee (q \vee r)$   $\vee e$  2, 3-4, 5-7
9.  $r$   $\vee e$  (2)  
[ $r$ ]  
 $p \vee (q \vee r)$
- 10.

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.	$(p \vee q) \vee r$	premise
2.	$p \vee q$	$\vee e\ (1)$
3.	$p$	$\vee e\ (1.1)$
4.	$p \vee (q \vee r)$	$\vee i_1\ 3$
5.	$q$	$\vee e\ (1.2)$
6.	$q \vee r$	$\vee i_1\ 5$
7.	$p \vee (q \vee r)$	$\vee i_2\ 6$
8.	$p \vee (q \vee r)$	$\vee e\ 2, 3-4, 5-7$
9.	$r$	$\vee e\ (2)$
10.	$q \vee r$	$\vee i_2\ 9$
11.		

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.  $q \vee r$   $\vee i_1$  5

7.  $p \vee (q \vee r)$   $\vee i_2$  6

8.  $p \vee (q \vee r)$   $\vee e$  2, 3-4, 5-7

9.  $r$   $\vee e$  (2)

10.  $q \vee r$   $\vee i_2$  9

11.  $p \vee (q \vee r)$   $\vee i_2$  10

# Associativity Using Or Elimination

►  $(p \vee q) \vee r \vdash p \vee (q \vee r)$

1.  $(p \vee q) \vee r$  premise

2.  $p \vee q$   $\vee e$  (1)

3.  $p$   $\vee e$  (1.1)

4.  $p \vee (q \vee r)$   $\vee i_1$  3

5.  $q$   $\vee e$  (1.2)

6.  $q \vee r$   $\vee i_1$  5

7.  $p \vee (q \vee r)$   $\vee i_2$  6

8.  $p \vee (q \vee r)$   $\vee e$  2, 3-4, 5-7

9.  $r$   $\vee e$  (2)

10.  $q \vee r$   $\vee i_2$  9

11.  $p \vee (q \vee r)$   $\vee i_2$  10

12.  $p \vee (q \vee r)$   $\vee e$  1, 2-8, 9-11

# Basic Rules So Far

---

- ▶  $\wedge i$ ,  $\wedge e_1$ ,  $\wedge e_2$  (and introduction and elimination)
- ▶  $\neg\neg e$ ,  $\neg\neg i$  (double negation elimination and introduction)
- ▶ MP (Modus Ponens)
- ▶  $\rightarrow i$  (Implies Introduction : remember opening boxes)
- ▶  $\vee i_1$ ,  $\vee i_2$ ,  $\vee e$  (Or introduction and elimination)

# The Copy Rule

---

►  $\vdash p \rightarrow (q \rightarrow p)$

1.     *true*                      premise
- 2.

# The Copy Rule

---

►  $\vdash p \rightarrow (q \rightarrow p)$

1. *true* premise
2.  $p$  assumption
- 3.

# The Copy Rule

---

- ▶  $\vdash p \rightarrow (q \rightarrow p)$

1.	<i>true</i>	premise
2.	<i>p</i>	assumption
3.	<i>q</i>	assumption
4.		

# The Copy Rule

---

►  $\vdash p \rightarrow (q \rightarrow p)$

1.	<i>true</i>	premise
2.	<i>p</i>	assumption
3.	<i>q</i>	assumption
4.	<i>p</i>	copy 2
5.		

# The Copy Rule

---

►  $\vdash p \rightarrow (q \rightarrow p)$

1.  $true$  premise
2.  $p$  assumption
3.  $q$  assumption
4.  $p$  copy 2
5.  $q \rightarrow p$   $\rightarrow i$  3-4
- 6.

# The Copy Rule

---

►  $\vdash p \rightarrow (q \rightarrow p)$

1.	<i>true</i>	premise
2.	<i>p</i>	assumption
3.	<i>q</i>	assumption
4.	<i>p</i>	copy 2
5.	$q \rightarrow p$	$\rightarrow i$ 3-4
6.	$p \rightarrow (q \rightarrow p)$	$\rightarrow i$ 2-5

# The Rules of Single Negation

---

- ▶ We have seen  $\neg\neg e$  and  $\neg\neg i$ , the elimination and introduction of double negation.

# The Rules of Single Negation

---

- ▶ We have seen  $\neg\neg e$  and  $\neg\neg i$ , the elimination and introduction of double negation.
- ▶ How about introducing and eliminating single negations?

# The Rules of Single Negation

---

- ▶ We have seen  $\neg\neg e$  and  $\neg\neg i$ , the elimination and introduction of double negation.
- ▶ How about introducing and eliminating single negations?
- ▶ We use the notion of **contradictions**, an expression of the form  $\varphi \wedge \neg\varphi$ , where  $\varphi$  is any propositional logic formula.

# The Rules of Single Negation

---

- ▶ We have seen  $\neg\neg e$  and  $\neg\neg i$ , the elimination and introduction of double negation.
- ▶ How about introducing and eliminating single negations?
- ▶ We use the notion of **contradictions**, an expression of the form  $\varphi \wedge \neg\varphi$ , where  $\varphi$  is any propositional logic formula.
- ▶ Any two contradictions are equivalent :  $p \wedge \neg p$  is equivalent to  $\neg r \wedge r$ . Contradictions denoted by  $\perp$ .

# The Rules of Single Negation

---

- ▶ We have seen  $\neg\neg e$  and  $\neg\neg i$ , the elimination and introduction of double negation.
- ▶ How about introducing and eliminating single negations?
- ▶ We use the notion of **contradictions**, an expression of the form  $\varphi \wedge \neg\varphi$ , where  $\varphi$  is any propositional logic formula.
- ▶ Any two contradictions are equivalent :  $p \wedge \neg p$  is equivalent to  $\neg r \wedge r$ . Contradictions denoted by  $\perp$ .
- ▶  $\perp \rightarrow \varphi$  for any formula  $\varphi$ .

# Rules with $\perp$

---

The  $\perp$  elimination rule  $\perp e$

$$\frac{\perp}{\psi}$$

The  $\perp$  introduction rule  $\perp i$

$$\frac{\varphi \quad \neg\varphi}{\perp}$$

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
- 2.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise

2.  $\boxed{\neg p \quad \vee e(1)}$

3.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
2.  $\boxed{\begin{array}{c} \neg p \quad \vee \text{e (1)} \\ p \quad \text{assumption} \end{array}}$
- 3.
- 4.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
2.  $\neg p \quad \vee e (1)$
3.  $p$  assumption
4.  $\perp \quad \perp i 2,3$
- 5.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
2.  $\neg p$   $\vee e (1)$
3.  $p$  assumption
4.  $\perp$   $\perp i 2,3$
5.  $q$   $\perp e 4$
- 6.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
2.  $\neg p$   $\vee e (1)$
3.  $p$  assumption
4.  $\perp$   $\perp i 2,3$
5.  $q$   $\perp e 4$
6.  $p \rightarrow q$   $\rightarrow i 3-5$
7.  $q$   $\vee e (2)$
- 8.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise

2.  $\neg p$   $\vee e$  (1)

3.  $p$  assumption

4.  $\perp$   $\perp i$  2,3

5.  $q$   $\perp e$  4

6.  $p \rightarrow q$   $\rightarrow i$  3-5

7.  $q$   $\vee e$  (2)

8.  $p$  assumption

9.

# An Example

---

►  $\neg p \vee q \vdash p \rightarrow q$

1.  $\neg p \vee q$  premise
2.  $\neg p$   $\vee e$  (1)
3.  $p$  assumption
4.  $\perp$   $\perp i$  2,3
5.  $q$   $\perp e$  4
6.  $p \rightarrow q$   $\rightarrow i$  3-5
7.  $q$   $\vee e$  (2)
8.  $p$  assumption
9.  $q$  copy 7
10.  $p \rightarrow q$   $\rightarrow i$  8-9
11.  $p \rightarrow q$   $\vee e$  1, 2-6, 7-10

# Introducing Negations (PBC)

---

- ▶ In the course of a proof, if you assume  $\varphi$  (by opening a box) and obtain  $\perp$  in the box, then we conclude  $\neg\varphi$
- ▶ This rule is denoted  $\neg i$  and is read as  $\neg$  introduction.
- ▶ Also known as Proof By Contradiction

# An Example

---

►  $p \rightarrow \neg p \vdash \neg p$

1.  $p \rightarrow \neg p$  premise

2.

# An Example

---

►  $p \rightarrow \neg p \vdash \neg p$

1.  $p \rightarrow \neg p$  premise
2.  $p$  assumption
- 3.

# An Example

---

►  $p \rightarrow \neg p \vdash \neg p$

1.  $p \rightarrow \neg p$  premise
2.  $p$  assumption
3.  $\neg p$  MP 1,2
- 4.

# An Example

---

►  $p \rightarrow \neg p \vdash \neg p$

1.  $p \rightarrow \neg p$  premise
2.  $p$  assumption
3.  $\neg p$  MP 1,2
4.  $\perp$   $\perp i$  2,3
5.  $\neg p$   $\neg i$  2-4

# The Last One

---

Law of the Excluded Middle (LEM)

$$\frac{}{\varphi \vee \neg\varphi}$$

# Summary of Basic Rules

---

- ▶  $\wedge i, \wedge e_1, \wedge e_2,$
- ▶  $\neg\neg e$
- ▶ MP
- ▶  $\rightarrow i$
- ▶  $\vee i_1, \vee i_2, \vee e$
- ▶ Copy,  $\neg i$  or PBC
- ▶  $\perp e, \perp i$

# Derived Rules

---

- ▶ MT (derive using MP,  $\perp i$  and  $\neg i$ )
- ▶  $\neg\neg i$  (derive using  $\perp i$  and  $\neg i$ )
- ▶ LEM (derive using  $\vee i_1$ ,  $\perp i$ ,  $\neg i$ ,  $\vee i_2$ ,  $\neg\neg e$ )

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached
- ▶ Intuitively,  $p \rightarrow q \vdash \neg p \vee q$  makes sense because you think semantically. However, we never used any semantics so far.

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached
- ▶ Intuitively,  $p \rightarrow q \vdash \neg p \vee q$  makes sense because you think semantically. However, we never used any semantics so far.
- ▶ Now we show that whatever can be proved makes sense semantically too.

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. **Truth tables** for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. Truth tables for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as semantically entails
  - ▶ Recall  $\vdash$ , and compare with  $\models$

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. **Truth tables** for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as **semantically entails**
  - ▶ Recall  $\vdash$ , and compare with  $\models$
- ▶ Formulae  $\varphi$  and  $\psi$  are **provably equivalent** iff  $\varphi \vdash \psi$  and  $\psi \vdash \varphi$

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. **Truth tables** for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as **semantically entails**
  - ▶ Recall  $\vdash$ , and compare with  $\models$
- ▶ Formulae  $\varphi$  and  $\psi$  are **provably equivalent** iff  $\varphi \vdash \psi$  and  $\psi \vdash \varphi$
- ▶ Formulae  $\varphi$  and  $\psi$  are **semantically equivalent** iff  $\varphi \models \psi$  and  $\psi \models \varphi$

# CS 228 : Logic in Computer Science

Krishna. S

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached
- ▶ Intuitively,  $p \rightarrow q \vdash \neg p \vee q$  makes sense because you think semantically. However, we never used any semantics so far.

# The Proofs So Far

---

- ▶ So far, the “proof” we have seen is purely syntactic, no true/false meanings were attached
- ▶ Intuitively,  $p \rightarrow q \vdash \neg p \vee q$  makes sense because you think semantically. However, we never used any semantics so far.
- ▶ Now we show that whatever can be proved makes sense semantically too.

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. Truth tables for each of the operators  $\vee$ ,  $\wedge$ ,  $\neg$ ,  $\rightarrow$  to determine truth values of complex formulae.

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. Truth tables for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as **semantically entails**

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. Truth tables for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as **semantically entails**
- ▶ Two formulae  $\varphi$  and  $\psi$  are **provably equivalent** iff  $\varphi \vdash \psi$  and  $\psi \vdash \varphi$

# Semantics

---

- ▶ Each propositional variable is assigned values true/false. Truth tables for each of the operators  $\vee, \wedge, \neg, \rightarrow$  to determine truth values of complex formulae.
- ▶  $\varphi_1, \dots, \varphi_n \models \psi$  iff whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .  $\models$  is read as **semantically entails**
- ▶ Two formulae  $\varphi$  and  $\psi$  are **provably equivalent** iff  $\varphi \vdash \psi$  and  $\psi \vdash \varphi$
- ▶ Two formulae  $\varphi$  and  $\psi$  are **semantically equivalent** iff  $\varphi \models \psi$  and  $\psi \models \varphi$

# Soundness of Propositional Logic

---

$$\varphi_1, \dots, \varphi_n \vdash \psi \Rightarrow \varphi_1, \dots, \varphi_n \vDash \psi$$

Whenever  $\psi$  can be proved from  $\varphi_1, \dots, \varphi_n$ , then  $\psi$  evaluates to true whenever  $\varphi_1, \dots, \varphi_n$  evaluate to true

# Soundness

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \vdash \psi$ .

# Soundness

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \vdash \psi$ .
- ▶ There is some proof (of length  $k$  lines) that yields  $\psi$ . Induct on  $k$ .

# Soundness

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \vdash \psi$ .
- ▶ There is some proof (of length  $k$  lines) that yields  $\psi$ . Induct on  $k$ .
- ▶ When  $k = 1$ , there is only one line in the proof, say  $\varphi$ , which is the premise. Then we have  $\varphi \vdash \varphi$ , since  $\varphi$  is given. But then we also have  $\varphi \vDash \varphi$ .

# Soundness

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \vdash \psi$ .
- ▶ There is some proof (of length  $k$  lines) that yields  $\psi$ . Induct on  $k$ .
- ▶ When  $k = 1$ , there is only one line in the proof, say  $\varphi$ , which is the premise. Then we have  $\varphi \vdash \varphi$ , since  $\varphi$  is given. But then we also have  $\varphi \models \varphi$ .
- ▶ Assume that whenever  $\varphi_1, \dots, \varphi_n \vdash \psi$  using a proof of length  $\leq k - 1$ , we have  $\varphi_1, \dots, \varphi_n \models \psi$ .

# Soundness

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \vdash \psi$ .
- ▶ There is some proof (of length  $k$  lines) that yields  $\psi$ . Induct on  $k$ .
- ▶ When  $k = 1$ , there is only one line in the proof, say  $\varphi$ , which is the premise. Then we have  $\varphi \vdash \varphi$ , since  $\varphi$  is given. But then we also have  $\varphi \vDash \varphi$ .
- ▶ Assume that whenever  $\varphi_1, \dots, \varphi_n \vdash \psi$  using a proof of length  $\leq k - 1$ , we have  $\varphi_1, \dots, \varphi_n \vDash \psi$ .
- ▶ Consider now a proof with  $k$  lines.

## Soundness : Case $\wedge i$

---

- ▶ How did we arrive at  $\psi$ ? Which proof rule gave  $\psi$  as the last line?

## Soundness : Case $\wedge i$

---

- ▶ How did we arrive at  $\psi$ ? Which proof rule gave  $\psi$  as the last line?
- ▶ Assume  $\psi$  was obtained using  $\wedge i$ . Then  $\psi$  is of the form  $\psi_1 \wedge \psi_2$ .

## Soundness : Case $\wedge i$

---

- ▶ How did we arrive at  $\psi$ ? Which proof rule gave  $\psi$  as the last line?
- ▶ Assume  $\psi$  was obtained using  $\wedge i$ . Then  $\psi$  is of the form  $\psi_1 \wedge \psi_2$ .
- ▶  $\psi_1$  and  $\psi_2$  were proved earlier, say in lines  $k_1, k_2 < k$ .

## Soundness : Case $\wedge i$

---

- ▶ How did we arrive at  $\psi$ ? Which proof rule gave  $\psi$  as the last line?
- ▶ Assume  $\psi$  was obtained using  $\wedge i$ . Then  $\psi$  is of the form  $\psi_1 \wedge \psi_2$ .
- ▶  $\psi_1$  and  $\psi_2$  were proved earlier, say in lines  $k_1, k_2 < k$ .
- ▶ We have the shorter proofs  $\varphi_1, \dots, \varphi_n \vdash \psi_1$  and  $\varphi_1, \dots, \varphi_n \vdash \psi_2$

## Soundness : Case $\wedge i$

---

- ▶ How did we arrive at  $\psi$ ? Which proof rule gave  $\psi$  as the last line?
- ▶ Assume  $\psi$  was obtained using  $\wedge i$ . Then  $\psi$  is of the form  $\psi_1 \wedge \psi_2$ .
- ▶  $\psi_1$  and  $\psi_2$  were proved earlier, say in lines  $k_1, k_2 < k$ .
- ▶ We have the shorter proofs  $\varphi_1, \dots, \varphi_n \vdash \psi_1$  and  $\varphi_1, \dots, \varphi_n \vdash \psi_2$
- ▶ By inductive hypothesis, we have  $\varphi_1, \dots, \varphi_n \models \psi_1$  and  $\varphi_1, \dots, \varphi_n \models \psi_2$ . By semantics, we have  $\varphi_1, \dots, \varphi_n \models \psi_1 \wedge \psi_2$ .

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .
- ▶ A box starting with  $\psi_1$  was opened at some line  $k_1 < k$ .

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .
- ▶ A box starting with  $\psi_1$  was opened at some line  $k_1 < k$ .
- ▶ The last line in the box was  $\psi_2$ .

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .
- ▶ A box starting with  $\psi_1$  was opened at some line  $k_1 < k$ .
- ▶ The last line in the box was  $\psi_2$ .
- ▶ The line just after the box was  $\psi$ .

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .
- ▶ A box starting with  $\psi_1$  was opened at some line  $k_1 < k$ .
- ▶ The last line in the box was  $\psi_2$ .
- ▶ The line just after the box was  $\psi$ .
- ▶ Consider adding  $\psi_1$  in the premises along with  $\varphi_1, \dots, \varphi_n$ . Then we will get a proof  $\varphi_1, \dots, \varphi_n, \psi_1 \vdash \psi_2$ , of length  $k - 1$ . By inductive hypothesis,  $\varphi_1, \dots, \varphi_n, \psi_1 \models \psi_2$ . By semantics, this is same as  $\varphi_1, \dots, \varphi_n \models \psi_1 \rightarrow \psi_2$

## Soundness : Case $\rightarrow i$

---

- ▶ Assume  $\psi$  was obtained using  $\rightarrow i$ . Then  $\psi$  is of the form  $\psi_1 \rightarrow \psi_2$ .
- ▶ A box starting with  $\psi_1$  was opened at some line  $k_1 < k$ .
- ▶ The last line in the box was  $\psi_2$ .
- ▶ The line just after the box was  $\psi$ .
- ▶ Consider adding  $\psi_1$  in the premises along with  $\varphi_1, \dots, \varphi_n$ . Then we will get a proof  $\varphi_1, \dots, \varphi_n, \psi_1 \vdash \psi_2$ , of length  $k - 1$ . By inductive hypothesis,  $\varphi_1, \dots, \varphi_n, \psi_1 \models \psi_2$ . By semantics, this is same as  $\varphi_1, \dots, \varphi_n \models \psi_1 \rightarrow \psi_2$
- ▶ The equivalence of  $\varphi_1, \dots, \varphi_n \vdash \psi_1 \rightarrow \psi_2$  and  $\varphi_1, \dots, \varphi_n, \psi_1 \vdash \psi_2$  gives the proof.

# Soundness : Other cases

---

# Completeness

---

$$\varphi_1, \dots, \varphi_n \models \psi \Rightarrow \varphi_1, \dots, \varphi_n \vdash \psi$$

Whenever  $\varphi_1, \dots, \varphi_n$  semantically entail  $\psi$ , then  $\psi$  can be proved from  $\varphi_1, \dots, \varphi_n$ . The proof rules are **complete**

# Completeness : 3 steps

---

- ▶ Given  $\varphi_1, \dots, \varphi_n \models \psi$

# Completeness : 3 steps

---

- ▶ Given  $\varphi_1, \dots, \varphi_n \models \psi$
- ▶ Step 1: Show that  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$

# Completeness : 3 steps

---

- ▶ Given  $\varphi_1, \dots, \varphi_n \models \psi$
- ▶ Step 1: Show that  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$
- ▶ Step 2: Show that  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$

# Completeness : 3 steps

---

- ▶ Given  $\varphi_1, \dots, \varphi_n \models \psi$
- ▶ Step 1: Show that  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$
- ▶ Step 2: Show that  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$
- ▶ Step 3: Show that  $\varphi_1, \dots, \varphi_n \vdash \psi$

# Completeness : Step 1

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \models \psi$ . Whenever all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .

# Completeness : Step 1

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \models \psi$ . Whenever all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .
- ▶ If  $\nvdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , then  $\psi$  evaluates to false when all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, a contradiction.

# Completeness : Step 1

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \models \psi$ . Whenever all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .
- ▶ If  $\not\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , then  $\psi$  evaluates to false when all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, a contradiction.
- ▶ Hence,  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .

## Completeness : Step 2

---

- ▶ Given  $\models \psi$ , show that  $\vdash \psi$

## Completeness : Step 2

---

- ▶ Given  $\models \psi$ , show that  $\vdash \psi$
- ▶ Assume  $p_1, \dots, p_n$  are the propositional variables in  $\psi$ . We know that all the  $2^n$  assignments of values to  $p_1, \dots, p_n$  make  $\psi$  true.

## Completeness : Step 2

---

- ▶ Given  $\models \psi$ , show that  $\vdash \psi$
- ▶ Assume  $p_1, \dots, p_n$  are the propositional variables in  $\psi$ . We know that all the  $2^n$  assignments of values to  $p_1, \dots, p_n$  make  $\psi$  true.
- ▶ Using this insight, we have to give a proof of  $\psi$ .

# Completeness : Step 2

---

## Truth Table to Proof

Let  $\varphi$  be a formula with variables  $p_1, \dots, p_n$ . Let  $\mathcal{T}$  be the truth table of  $\varphi$ , and let  $l$  be a line number in  $\mathcal{T}$ . Let  $\hat{p}_i$  represent  $p_i$  if  $p_i$  is assigned true in line  $l$ , and let it denote  $\neg p_i$  if  $p_i$  is assigned false in line  $l$ . Then

1.  $\hat{p}_1, \dots, \hat{p}_n \vdash \varphi$  if  $\varphi$  evaluates to true in line  $l$
2.  $\hat{p}_1, \dots, \hat{p}_n \vdash \neg\varphi$  if  $\varphi$  evaluates to false in line  $l$

# CS 228 : Logic in Computer Science

Krishna. S

# Completeness

---

$$\varphi_1, \dots, \varphi_n \models \psi \Rightarrow \varphi_1, \dots, \varphi_n \vdash \psi$$

Whenever  $\varphi_1, \dots, \varphi_n$  semantically entail  $\psi$ , then  $\psi$  can be proved from  $\varphi_1, \dots, \varphi_n$ . The proof rules are **complete**

# Completeness : 3 steps

---

- ▶ Given  $\varphi_1, \dots, \varphi_n \models \psi$
- ▶ Step 1: Show that  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$
- ▶ Step 2: Show that  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$
- ▶ Step 3: Show that  $\varphi_1, \dots, \varphi_n \vdash \psi$

# Completeness : Step 1

---

- ▶ Assume  $\varphi_1, \dots, \varphi_n \models \psi$ . Whenever all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, so does  $\psi$ .
- ▶ If  $\not\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , then  $\psi$  evaluates to false when all of  $\varphi_1, \dots, \varphi_n$  evaluate to true, a contradiction.
- ▶ Hence,  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .

# Completeness : Step 2

---

- ▶ Given  $\models \psi$ , show that  $\vdash \psi$
- ▶ Assume  $p_1, \dots, p_n$  are the propositional variables in  $\psi$ . We know that all the  $2^n$  assignments of values to  $p_1, \dots, p_n$  make  $\psi$  true.
- ▶ Using this insight, we have to give a proof of  $\psi$ .

# Completeness : Step 2

---

## Truth Table to Proof

Let  $\varphi$  be a formula with variables  $p_1, \dots, p_n$ . Let  $\mathcal{T}$  be the truth table of  $\varphi$ , and let  $l$  be a line number in  $\mathcal{T}$ . Let  $\hat{p}_i$  represent  $p_i$  if  $p_i$  is assigned true in line  $l$ , and let it denote  $\neg p_i$  if  $p_i$  is assigned false in line  $l$ . Then

1.  $\hat{p}_1, \dots, \hat{p}_n \vdash \varphi$  if  $\varphi$  evaluates to true in line  $l$
2.  $\hat{p}_1, \dots, \hat{p}_n \vdash \neg\varphi$  if  $\varphi$  evaluates to false in line  $l$

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .
- ▶ Base : If  $\varphi = p$ , a proposition, then we have  $p \vdash p$  and  $\neg p \vdash \neg p$ .

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .
- ▶ Base : If  $\varphi = p$ , a proposition, then we have  $p \vdash p$  and  $\neg p \vdash \neg p$ .
- ▶ Assume for formulae of size  $\leq k - 1$  (size=height of the parse tree). **What is a parse tree?**

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .
- ▶ Base : If  $\varphi = p$ , a proposition, then we have  $p \vdash p$  and  $\neg p \vdash \neg p$ .
- ▶ Assume for formulae of size  $\leq k - 1$  (size=height of the parse tree). **What is a parse tree?**
- ▶ Case Negation :  $\varphi = \neg\varphi_1$

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .
- ▶ Base : If  $\varphi = p$ , a proposition, then we have  $p \vdash p$  and  $\neg p \vdash \neg p$ .
- ▶ Assume for formulae of size  $\leq k - 1$  (size=height of the parse tree). **What is a parse tree?**
- ▶ Case Negation :  $\varphi = \neg\varphi_1$ 
  - ▶ Assume  $\varphi$  evaluates to true in line  $l$  of  $\mathcal{T}$ . Then  $\varphi_1$  evaluates to false in line  $l$ . By inductive hypothesis,  $\hat{p}_1, \dots, \hat{p}_n \vdash \neg\varphi_1$ .

# Truth Table to Proof

---

- ▶ Structural Induction on  $\varphi$ .
- ▶ Base : If  $\varphi = p$ , a proposition, then we have  $p \vdash p$  and  $\neg p \vdash \neg p$ .
- ▶ Assume for formulae of size  $\leq k - 1$  (size=height of the parse tree). **What is a parse tree?**
- ▶ Case Negation :  $\varphi = \neg\varphi_1$ 
  - ▶ Assume  $\varphi$  evaluates to true in line  $l$  of  $\mathcal{T}$ . Then  $\varphi_1$  evaluates to false in line  $l$ . By inductive hypothesis,  $\hat{p}_1, \dots, \hat{p}_n \vdash \neg\varphi_1$ .
  - ▶ Assume  $\varphi$  evaluates to false in line  $l$  of  $\mathcal{T}$ . Then  $\varphi_1$  evaluates to true in line  $l$ . By inductive hypothesis,  $\hat{p}_1, \dots, \hat{p}_n \vdash \varphi_1$ . Use the  $\neg\neg i$  rule to obtain a proof of  $\hat{p}_1, \dots, \hat{p}_n \vdash \neg\neg\varphi_1$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to false in line  $l$ , then  $\varphi_1$  evaluates to true and  $\varphi_2$  to false. Let  $\{q_1, \dots, q_k\}$  be the variables of  $\varphi_1$  and let  $\{r_1, \dots, r_j\}$  be the variables in  $\varphi_2$ .  $\{q_1, \dots, q_k\} \cup \{r_1, \dots, r_j\} = \{p_1, \dots, p_n\}$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to false in line  $l$ , then  $\varphi_1$  evaluates to true and  $\varphi_2$  to false. Let  $\{q_1, \dots, q_k\}$  be the variables of  $\varphi_1$  and let  $\{r_1, \dots, r_j\}$  be the variables in  $\varphi_2$ .  $\{q_1, \dots, q_k\} \cup \{r_1, \dots, r_j\} = \{p_1, \dots, p_n\}$ .
  - ▶ By inductive hypothesis,  $\hat{q}_1, \dots, \hat{q}_k \models \varphi_1$  and  $\hat{r}_1, \dots, \hat{r}_j \models \neg\varphi_2$ . Then,  $\hat{p}_1, \dots, \hat{p}_n \models \varphi_1 \wedge \neg\varphi_2$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to false in line  $l$ , then  $\varphi_1$  evaluates to true and  $\varphi_2$  to false. Let  $\{q_1, \dots, q_k\}$  be the variables of  $\varphi_1$  and let  $\{r_1, \dots, r_j\}$  be the variables in  $\varphi_2$ .  $\{q_1, \dots, q_k\} \cup \{r_1, \dots, r_j\} = \{p_1, \dots, p_n\}$ .
  - ▶ By inductive hypothesis,  $\hat{q}_1, \dots, \hat{q}_k \models \varphi_1$  and  $\hat{r}_1, \dots, \hat{r}_j \models \neg\varphi_2$ . Then,  $\hat{p}_1, \dots, \hat{p}_n \models \varphi_1 \wedge \neg\varphi_2$ .
  - ▶ Prove that  $\varphi_1 \wedge \neg\varphi_2 \vdash \neg(\varphi_1 \rightarrow \varphi_2)$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to true in line  $l$ , then there are 3 possibilities. If both  $\varphi_1, \varphi_2$  evaluate to true, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \varphi_1 \wedge \varphi_2$ . Proving  $\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to true in line  $l$ , then there are 3 possibilities. If both  $\varphi_1, \varphi_2$  evaluate to true, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \varphi_1 \wedge \varphi_2$ . Proving  $\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.
  - ▶ If both  $\varphi_1, \varphi_2$  evaluate to false, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \neg\varphi_1 \wedge \neg\varphi_2$ . Proving  $\neg\varphi_1 \wedge \neg\varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.

# Truth Table to Proof

---

- ▶ Case  $\rightarrow$  :  $\varphi = \varphi_1 \rightarrow \varphi_2$ .
  - ▶ If  $\varphi$  evaluates to true in line  $l$ , then there are 3 possibilities. If both  $\varphi_1, \varphi_2$  evaluate to true, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \varphi_1 \wedge \varphi_2$ . Proving  $\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.
  - ▶ If both  $\varphi_1, \varphi_2$  evaluate to false, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \neg\varphi_1 \wedge \neg\varphi_2$ . Proving  $\neg\varphi_1 \wedge \neg\varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.
  - ▶ Last, if  $\varphi_1$  evaluates to false and  $\varphi_2$  evaluates to true, then we have  $\hat{p}_1, \dots, \hat{p}_n \models \neg\varphi_1 \wedge \varphi_2$ . Proving  $\neg\varphi_1 \wedge \varphi_2 \vdash \varphi_1 \rightarrow \varphi_2$ , we are done.

# Truth Table to Proof

---

- ▶ Prove the cases  $\wedge, \vee$ .

# On An Example

---

We know  $\models (p \wedge q) \rightarrow p$ . Using this fact, show that  $\vdash (p \wedge q) \rightarrow p$ .

- ▶  $p, q \vdash (p \wedge q) \rightarrow p$
- ▶  $\neg p, q \vdash (p \wedge q) \rightarrow p$
- ▶  $p, \neg q \vdash (p \wedge q) \rightarrow p$
- ▶  $\neg p, \neg q \vdash (p \wedge q) \rightarrow p$

Now, combine the 4 proofs above to give a single proof for  
 $\vdash (p \wedge q) \rightarrow p$ .

# Completeness : Steps 2, 3

---

- ▶ Step 2: From  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , use LEM on all the propositional variables of  $\varphi_1, \dots, \varphi_n, \psi$  to obtain  
 $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .

# Completeness : Steps 2, 3

---

- ▶ Step 2: From  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , use LEM on all the propositional variables of  $\varphi_1, \dots, \varphi_n, \psi$  to obtain  
 $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .
- ▶ Step 3: Take the proof  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ . This proof has  $n$  nested boxes, the  $i$ th box opening with the assumption  $\varphi_i$ . The last box closes with the last line  $\psi$ . Hence, the line immediately after the last box is  $\varphi_n \rightarrow \psi$ .

# Completeness : Steps 2, 3

---

- ▶ Step 2: From  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , use LEM on all the propositional variables of  $\varphi_1, \dots, \varphi_n, \psi$  to obtain  
 $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .
- ▶ Step 3: Take the proof  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ . This proof has  $n$  nested boxes, the  $i$ th box opening with the assumption  $\varphi_i$ . The last box closes with the last line  $\psi$ . Hence, the line immediately after the last box is  $\varphi_n \rightarrow \psi$ .
- ▶ In a similar way, the  $(n - 1)$ th box has as its last line  $\varphi_n \rightarrow \psi$ , and hence, the line immediately after this box is  $\varphi_{n-1} \rightarrow (\varphi_n \rightarrow \psi)$  and so on.

# Completeness : Steps 2, 3

---

- ▶ Step 2: From  $\models \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ , use LEM on all the propositional variables of  $\varphi_1, \dots, \varphi_n, \psi$  to obtain  
 $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ .
- ▶ Step 3: Take the proof  $\vdash \varphi_1 \rightarrow (\varphi_2 \rightarrow (\dots (\varphi_n \rightarrow \psi) \dots))$ . This proof has  $n$  nested boxes, the  $i$ th box opening with the assumption  $\varphi_i$ . The last box closes with the last line  $\psi$ . Hence, the line immediately after the last box is  $\varphi_n \rightarrow \psi$ .
- ▶ In a similar way, the  $(n - 1)$ th box has as its last line  $\varphi_n \rightarrow \psi$ , and hence, the line immediately after this box is  $\varphi_{n-1} \rightarrow (\varphi_n \rightarrow \psi)$  and so on.
- ▶ Add premises  $\varphi_1, \dots, \varphi_n$  on the top. Use MP on the premises, and the lines after boxes 1 to  $n$  in order to obtain  $\psi$ .

# Summary

---

Propositional Logic is sound and complete.

# Normal Forms

---

- ▶ A **literal** is a propositional variable  $p$  or its negation  $\neg p$ . These are referred to as positive and negative literals respectively.

# Normal Forms

---

- ▶ A **literal** is a propositional variable  $p$  or its negation  $\neg p$ . These are referred to as positive and negative literals respectively.
- ▶ A formula  $F$  is in CNF if it is a conjunction of a disjunction of literals.

$$F = \bigwedge_{i=1}^n \bigvee_{j=1}^m L_{i,j}$$

each  $L_{i,j}$  is a literal.

# Normal Forms

---

- ▶ A **literal** is a propositional variable  $p$  or its negation  $\neg p$ . These are referred to as positive and negative literals respectively.
- ▶ A formula  $F$  is in CNF if it is a conjunction of a disjunction of literals.

$$F = \bigwedge_{i=1}^n \bigvee_{j=1}^m L_{i,j}$$

each  $L_{i,j}$  is a literal.

- ▶ A formula  $F$  is in DNF if it is a disjunction of a conjunction of literals.

$$F = \bigvee_{i=1}^n \bigwedge_{j=1}^m L_{i,j}$$

each  $L_{i,j}$  is a literal.

# Normal Forms

---

In the following, equivalent stands for semantically equivalent

Let  $F$  be a formula in CNF and let  $G$  be a formula in DNF. Then  $\neg F$  is equivalent to a formula in DNF and  $\neg G$  is equivalent to a formula in CNF.

# Normal Forms

---

In the following, equivalent stands for semantically equivalent

Let  $F$  be a formula in CNF and let  $G$  be a formula in DNF. Then  $\neg F$  is equivalent to a formula in DNF and  $\neg G$  is equivalent to a formula in CNF.

Every formula  $F$  is equivalent to some formula  $F_1$  in CNF and some formula  $F_2$  in DNF.

# CNF Algorithm

---

Given a formula  $F$ ,  $(x \rightarrow [\neg(y \vee z) \wedge \neg(y \rightarrow x)])$

- ▶ Replace all subformulae of the form  $F \rightarrow G$  with  $\neg F \vee G$ , and all subformulae of the form  $F \leftrightarrow G$  with  $(\neg F \vee G) \wedge (\neg G \vee F)$ . When there are no more occurrences of  $\rightarrow, \leftrightarrow$ , proceed to the next step.

# CNF Algorithm

---

Given a formula  $F$ ,  $(x \rightarrow [\neg(y \vee z) \wedge \neg(y \rightarrow x)])$

- ▶ Replace all subformulae of the form  $F \rightarrow G$  with  $\neg F \vee G$ , and all subformulae of the form  $F \leftrightarrow G$  with  $(\neg F \vee G) \wedge (\neg G \vee F)$ . When there are no more occurrences of  $\rightarrow, \leftrightarrow$ , proceed to the next step.
- ▶ Get rid of all double negations : Replace all subformulae
  - ▶  $\neg\neg G$  with  $G$ ,
  - ▶  $\neg(G \wedge H)$  with  $\neg G \vee \neg H$
  - ▶  $\neg(G \vee H)$  with  $\neg G \wedge \neg H$

When there are no more such subformulae, proceed to the next step.

# CNF Algorithm

---

Given a formula  $F$ ,  $(x \rightarrow [\neg(y \vee z) \wedge \neg(y \rightarrow x)])$

- ▶ Replace all subformulae of the form  $F \rightarrow G$  with  $\neg F \vee G$ , and all subformulae of the form  $F \leftrightarrow G$  with  $(\neg F \vee G) \wedge (\neg G \vee F)$ . When there are no more occurrences of  $\rightarrow, \leftrightarrow$ , proceed to the next step.
- ▶ Get rid of all double negations : Replace all subformulae
  - ▶  $\neg\neg G$  with  $G$ ,
  - ▶  $\neg(G \wedge H)$  with  $\neg G \vee \neg H$
  - ▶  $\neg(G \vee H)$  with  $\neg G \wedge \neg H$

When there are no more such subformulae, proceed to the next step.

- ▶ Distribute  $\vee$  wherever possible.

The resultant formula  $F_1$  is in CNF and is provably equivalent to  $F$ .

$$[(\neg x \vee \neg y) \wedge (\neg x \vee \neg z)] \wedge [(\neg x \vee y) \wedge (\neg x \vee \neg x)]$$

# The Hardness of SAT

---

- ▶ Given a formula  $\varphi$  how to check if  $\varphi$  is satisfiable?
- ▶ Given a formula  $\varphi$  how to check if  $\varphi$  is unsatisfiable?
- ▶ SAT is NP-complete

## Polynomial Time Formula Classes

# Horn Formulae

---

- ▶ A **Horn Formula** is a particularly nice kind of CNF formula, which can be **quickly** checked for satisfiability.
- ▶ Programming languages Prolog and Datalog are based on Horn clauses in first order logic

# Horn Formulae

---

- ▶ A **Horn Formula** is a particularly nice kind of CNF formula, which can be **quickly** checked for satisfiability.
- ▶ Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- ▶ A formula  $F$  is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.

# Horn Formulae

---

- ▶ A **Horn Formula** is a particularly nice kind of CNF formula, which can be **quickly** checked for satisfiability.
- ▶ Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- ▶ A formula  $F$  is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.
- ▶  $p \wedge (\neg p \vee \neg q \vee r) \wedge (\neg a \vee \neg b)$  is Horn, but  $a \vee b$  is not Horn.

# Horn Formulae

---

- ▶ A **Horn Formula** is a particularly nice kind of CNF formula, which can be **quickly** checked for satisfiability.
- ▶ Programming languages Prolog and Datalog are based on Horn clauses in first order logic
- ▶ A formula  $F$  is a Horn formula if it is in CNF and every disjunction contains atmost one positive literal.
- ▶  $p \wedge (\neg p \vee \neg q \vee r) \wedge (\neg a \vee \neg b)$  is Horn, but  $a \vee b$  is not Horn.
- ▶ A basic Horn formula is one which has no  $\wedge$ . Every Horn formula is a conjunction of basic Horn formulae.

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- ▶ Basic Horn with both positive and negative literals are written as an implication  $p \wedge q \wedge \cdots \wedge r \rightarrow s$  involving only positive literals.

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- ▶ Basic Horn with both positive and negative literals are written as an implication  $p \wedge q \wedge \cdots \wedge r \rightarrow s$  involving only positive literals.
- ▶ Basic Horn with no negative literals are of the form  $p$  and are written as  $\top \rightarrow p$ .

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- ▶ Basic Horn with both positive and negative literals are written as an implication  $p \wedge q \wedge \cdots \wedge r \rightarrow s$  involving only positive literals.
- ▶ Basic Horn with no negative literals are of the form  $p$  and are written as  $\top \rightarrow p$ .
- ▶ Basic Horn with no positive literals are written as  $p \wedge q \wedge \cdots \wedge r \rightarrow \perp$ .

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- ▶ Basic Horn with both positive and negative literals are written as an implication  $p \wedge q \wedge \cdots \wedge r \rightarrow s$  involving only positive literals.
- ▶ Basic Horn with no negative literals are of the form  $p$  and are written as  $\top \rightarrow p$ .
- ▶ Basic Horn with no positive literals are written as  $p \wedge q \wedge \cdots \wedge r \rightarrow \perp$ .
- ▶ Thus, a Horn formula is written as a conjunction of implications.

# CS 228 : Logic in Computer Science

Krishna. S

## Polynomial Time Formula Classes

# Horn Formulae

---

- ▶ A formula  $F$  is a Horn formula if it is in CNF and every disjunction contains at most one positive literal.
- ▶  $p \wedge (\neg p \vee \neg q \vee r) \wedge (\neg a \vee \neg b)$  is Horn, but  $a \vee b$  is not Horn.
- ▶ A basic Horn formula is one which has no  $\wedge$ . Every Horn formula is a conjunction of basic Horn formulae.

# Horn Formulae

---

- ▶ Three types of basic Horn : no positive literals, no negative literals, have both positive and negative literals.
- ▶ Basic Horn with both positive and negative literals are written as an implication  $p \wedge q \wedge \cdots \wedge r \rightarrow s$  involving only positive literals.
- ▶ Basic Horn with no negative literals are of the form  $p$  and are written as  $\top \rightarrow p$ .
- ▶ Basic Horn with no positive literals are written as  $p \wedge q \wedge \cdots \wedge r \rightarrow \perp$ .
- ▶ Thus, a Horn formula is written as a conjunction of implications.

# The Horn Algorithm

---

Given a Horn formula  $H$ ,

- ▶ Mark all occurrences of  $p$ , whenever  $\top \rightarrow p$  is a subformula.

# The Horn Algorithm

---

Given a Horn formula  $H$ ,

- ▶ Mark all occurrences of  $p$ , whenever  $\top \rightarrow p$  is a subformula.
- ▶ If there is a subformula of the form  $(p_1 \wedge \cdots \wedge p_m) \rightarrow q$ , where each  $p_i$  is marked, and  $q$  is not marked, mark  $q$ . Repeat this until there are no subformulae of this form and proceed to the next step.

# The Horn Algorithm

---

Given a Horn formula  $H$ ,

- ▶ Mark all occurrences of  $p$ , whenever  $\top \rightarrow p$  is a subformula.
- ▶ If there is a subformula of the form  $(p_1 \wedge \cdots \wedge p_m) \rightarrow q$ , where each  $p_i$  is marked, and  $q$  is not marked, mark  $q$ . Repeat this until there are no subformulae of this form and proceed to the next step.
- ▶ Consider subformulae of the form  $(p_1 \wedge \cdots \wedge p_m) \rightarrow \perp$ . If there is one such subformula with all  $p_i$  marked, then say **Unsat**, otherwise say **Sat**.

# An Example

---

$$(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B).$$

# An Example

---

$(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B).$

- $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B).$

# An Example

---

$$(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B).$$

- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .

# An Example

---

$(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .

- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .

# An Example

---

$$(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B).$$

- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .
- ▶  $(\top \rightarrow A) \wedge (C \rightarrow D) \wedge ((A \wedge B) \rightarrow C) \wedge ((C \wedge D) \rightarrow \perp) \wedge (\top \rightarrow B)$ .

# The Horn Algorithm

---

The Horn algorithm concludes Sat iff  $H$  is satisfiable.

# Complexity of Horn

---

- ▶ Given a Horn formula  $\psi$  with  $n$  propositions, how many times do you have to read  $\psi$ ?
- ▶ Step 1: Read once
- ▶ Step 2: Read almost  $n$  times
- ▶ Step 3: Read once

## 2-CNF

---

- ▶ 2-CNF : CNF where each clause has at most 2 literals.

# Resolution

---

- ▶ Resolution is a technique used to check if a formula in CNF is unsatisfiable.

# Resolution

---

- ▶ Resolution is a technique used to check if a formula in CNF is unsatisfiable.
- ▶ CNF notation as set of sets :  $(p \vee q) \wedge (\neg p \vee q) \wedge p$  represented as  $\{\{p, q\}, \{\neg p, q\}, \{p\}\}$

# Resolution

---

- ▶ Resolution is a technique used to check if a formula in CNF is unsatisfiable.
- ▶ CNF notation as set of sets :  $(p \vee q) \wedge (\neg p \vee q) \wedge p$  represented as  $\{\{p, q\}, \{\neg p, q\}, \{p\}\}$
- ▶ Let  $C_1, C_2$  be two clauses. Assume  $p \in C_1$  and  $\neg p \in C_2$  for some literal  $p$ . Then the clause  $R = (C_1 - \{p\}) \cup (C_2 - \{\neg p\})$  is a **resolvent** of  $C_1$  and  $C_2$ .

# Resolution

---

- ▶ Resolution is a technique used to check if a formula in CNF is unsatisfiable.
- ▶ CNF notation as set of sets :  $(p \vee q) \wedge (\neg p \vee q) \wedge p$  represented as  $\{\{p, q\}, \{\neg p, q\}, \{p\}\}$
- ▶ Let  $C_1, C_2$  be two clauses. Assume  $p \in C_1$  and  $\neg p \in C_2$  for some literal  $p$ . Then the clause  $R = (C_1 - \{p\}) \cup (C_2 - \{\neg p\})$  is a **resolvent** of  $C_1$  and  $C_2$ .
- ▶ Let  $C_1 = \{p_1, \neg p_2, p_3\}$  and  $C_2 = \{p_2, \neg p_3, p_4\}$ . As  $p_3 \in C_1$  and  $\neg p_3 \in C_2$ , we can find the resolvent. The resolvent is  $\{p_1, p_2, \neg p_2, p_4\}$ .

# Resolution

---

- ▶ Resolution is a technique used to check if a formula in CNF is unsatisfiable.
- ▶ CNF notation as set of sets :  $(p \vee q) \wedge (\neg p \vee q) \wedge p$  represented as  $\{\{p, q\}, \{\neg p, q\}, \{p\}\}$
- ▶ Let  $C_1, C_2$  be two clauses. Assume  $p \in C_1$  and  $\neg p \in C_2$  for some literal  $p$ . Then the clause  $R = (C_1 - \{p\}) \cup (C_2 - \{\neg p\})$  is a **resolvent** of  $C_1$  and  $C_2$ .
- ▶ Let  $C_1 = \{p_1, \neg p_2, p_3\}$  and  $C_2 = \{p_2, \neg p_3, p_4\}$ . As  $p_3 \in C_1$  and  $\neg p_3 \in C_2$ , we can find the resolvent. The resolvent is  $\{p_1, p_2, \neg p_2, p_4\}$ .
- ▶ Resolvent not unique :  $\{p_1, p_3, \neg p_3, p_4\}$  is also a resolvent.

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)
- ▶ Let  $F$  be a formula in CNF, and let  $C$  be a clause in  $F$ . Then  $F \vdash C$  (Prove!)

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)
- ▶ Let  $F$  be a formula in CNF, and let  $C$  be a clause in  $F$ . Then  $F \vdash C$  (Prove!)
- ▶ Let  $F$  be a formula in CNF. Let  $R$  be a resolvent of two clauses of  $F$ . Then  $F \vdash R$  (Prove!)

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$
- ▶  $\text{Res}^0(F) = F$ , there are finitely many clauses that can be derived from  $F$ .

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$
- ▶  $\text{Res}^0(F) = F$ , there are finitely many clauses that can be derived from  $F$ .
- ▶ There is some  $m \geq 0$  such that  $\text{Res}^m(F) = \text{Res}^{m+1}(F)$ . Denote it by  $\text{Res}^*(F)$ .

# Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$

# Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$
- ▶  $\text{Res}^1(F) = F \cup \{p_1, p_2, \neg p_2\} \cup \{p_1, \neg p_3, p_3\}$ .

# Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$
- ▶  $\text{Res}^1(F) = F \cup \{p_1, p_2, \neg p_2\} \cup \{p_1, \neg p_3, p_3\}$ .
- ▶  $\text{Res}^2(F) = \text{Res}^1(F) \cup \{p_1, p_2, \neg p_3\} \cup \{p_1, p_3, \neg p_2\}$

# **CS 228 : Logic in Computer Science**

Krishna. S

# Resolution

---

- ▶ Given a propositional logic formula  $\varphi$ , is it unsatisfiable?

# Resolution

---

- ▶ Given a propositional logic formula  $\varphi$ , is it unsatisfiable?
- ▶ How does a solver do it?
- ▶ Assume it is in CNF

# Resolution

---

- ▶ Let  $C_1, C_2$  be two clauses. Assume  $p \in C_1$  and  $\neg p \in C_2$  for some literal  $p$ .

# Resolution

---

- ▶ Let  $C_1, C_2$  be two clauses. Assume  $p \in C_1$  and  $\neg p \in C_2$  for some literal  $p$ . Then the clause  $R = (C_1 - \{p\}) \cup (C_2 - \{\neg p\})$  is a **resolvent** of  $C_1$  and  $C_2$ .
- ▶ Let  $C_1 = \{p_1, \neg p_2, p_3\}$  and  $C_2 = \{p_2, \neg p_3, p_4\}$ . As  $p_3 \in C_1$  and  $\neg p_3 \in C_2$ , we can find the resolvent. The resolvent is  $\{p_1, p_2, \neg p_2, p_4\}$ .
- ▶ Resolvent not unique :  $\{p_1, p_3, \neg p_3, p_4\}$  is also a resolvent.

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)
- ▶ Let  $F$  be a formula in CNF, and let  $C$  be a clause in  $F$ . Then  $F \vdash C$  (Prove!)

# 3 rules in Resolution

---

- ▶ Let  $G$  be any formula. Let  $F$  be the CNF formula resulting from the CNF algorithm applied to  $G$ . Then  $G \vdash F$  (Prove!)
- ▶ Let  $F$  be a formula in CNF, and let  $C$  be a clause in  $F$ . Then  $F \vdash C$  (Prove!)
- ▶ Let  $F$  be a formula in CNF. Let  $R$  be a resolvent of two clauses of  $F$ . Then  $F \vdash R$  (Prove!)

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$
- ▶  $\text{Res}^0(F) = F$ , there are finitely many clauses that can be derived from  $F$ .

# Completeness of Resolution

---

Show that resolution can be used to determine whether any given formula is unsatisfiable.

- ▶ Given  $F$  in CNF, let  $\text{Res}^0(F) = \{C \mid C \text{ is a clause in } F\}$ .
- ▶  $\text{Res}^n(F) = \text{Res}^{n-1}(F) \cup \{R \mid R \text{ is a resolvent of two clauses in } \text{Res}^{n-1}(F)\}$
- ▶  $\text{Res}^0(F) = F$ , there are finitely many clauses that can be derived from  $F$ .
- ▶ There is some  $m \geq 0$  such that  $\text{Res}^m(F) = \text{Res}^{m+1}(F)$ . Denote it by  $\text{Res}^*(F)$ .

## Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$

# Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$
- ▶  $\text{Res}^1(F) = F \cup \{p_1, p_2, \neg p_2\} \cup \{p_1, \neg p_3, p_3\}$ .

# Example

---

Let  $F = \{\{p_1, p_2, \neg p_3\}, \{\neg p_2, p_3\}\}$ .

- ▶  $\text{Res}^0(F) = F$
- ▶  $\text{Res}^1(F) = F \cup \{p_1, p_2, \neg p_2\} \cup \{p_1, \neg p_3, p_3\}$ .
- ▶  $\text{Res}^2(F) = \text{Res}^1(F) \cup \{p_1, p_2, \neg p_3\} \cup \{p_1, p_3, \neg p_2\}$

# Resolution

---

Let  $F$  be a formula in CNF. If  $\emptyset \in Res^*(F)$ , then  $F$  is unsatisfiable.

- ▶ If  $\emptyset \in Res^*(F)$ . Then  $\emptyset \in Res^n(F)$  for some  $n$ .

# Resolution

---

Let  $F$  be a formula in CNF. If  $\emptyset \in Res^*(F)$ , then  $F$  is unsatisfiable.

- ▶ If  $\emptyset \in Res^*(F)$ . Then  $\emptyset \in Res^n(F)$  for some  $n$ .
- ▶ Since  $\emptyset \notin Res^0(F)$  ( $\emptyset$  is not a clause), there is an  $m > 0$  such that  $\emptyset \notin Res^m(F)$  and  $\emptyset \in Res^{m+1}(F)$ .

# Resolution

---

Let  $F$  be a formula in CNF. If  $\emptyset \in Res^*(F)$ , then  $F$  is unsatisfiable.

- ▶ If  $\emptyset \in Res^*(F)$ . Then  $\emptyset \in Res^n(F)$  for some  $n$ .
- ▶ Since  $\emptyset \notin Res^0(F)$  ( $\emptyset$  is not a clause), there is an  $m > 0$  such that  $\emptyset \notin Res^m(F)$  and  $\emptyset \in Res^{m+1}(F)$ .
- ▶ Then  $\{p\}, \{\neg p\} \in Res^m(F)$ . By the rules of resolution, we have  $F \vdash p, \neg p$ , and thus  $F \vdash \perp$ . Hence,  $F$  is unsatisfiable.

# Resolution

---

Prove the converse:  $F$  is unsatisfiable implies  $\emptyset \in \text{Res}^*(F)$ .

(Discuss substitution before the proof)

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .
- ▶ wlg, assume that no  $C_i$  has both  $p$  and  $\neg p$

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .
- ▶ wlg, assume that no  $C_i$  has both  $p$  and  $\neg p$
- ▶ Induct on the number  $n$  of propositional variables that occur in  $F$ .

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .
- ▶ wlg, assume that no  $C_i$  has both  $p$  and  $\neg p$
- ▶ Induct on the number  $n$  of propositional variables that occur in  $F$ .
- ▶ If  $n = 1$ , then the possible clauses are  $p$ ,  $\neg p$  and  $p \vee \neg p$ . The third one is ruled out, by assumption.

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .
- ▶ wlg, assume that no  $C_i$  has both  $p$  and  $\neg p$
- ▶ Induct on the number  $n$  of propositional variables that occur in  $F$ .
- ▶ If  $n = 1$ , then the possible clauses are  $p$ ,  $\neg p$  and  $p \vee \neg p$ . The third one is ruled out, by assumption.
- ▶ If  $F = \{\{p\}\}$  or  $F = \{\{\neg p\}\}$ ,  $F$  is satisfiable.

# Resolution

---

If  $F$  in CNF is unsatisfiable, then  $\emptyset \in Res^*(F)$ .

- ▶ Let  $F$  have  $k$  clauses  $C_1, \dots, C_k$ .
- ▶ wlg, assume that no  $C_i$  has both  $p$  and  $\neg p$
- ▶ Induct on the number  $n$  of propositional variables that occur in  $F$ .
- ▶ If  $n = 1$ , then the possible clauses are  $p$ ,  $\neg p$  and  $p \vee \neg p$ . The third one is ruled out, by assumption.
- ▶ If  $F = \{\{p\}\}$  or  $F = \{\{\neg p\}\}$ ,  $F$  is satisfiable.
- ▶ Hence,  $F = \{\{p\}, \{\neg p\}\}$ . Clearly,  $\emptyset \in Res(F)$ .

# Resolution

---

- ▶ Inductive hypothesis : If  $F$  has  $\leq n$  variables and is unsat, then  $\emptyset \in \text{Res}^*(F)$ .

# Resolution

---

- ▶ Inductive hypothesis : If  $F$  has  $\leq n$  variables and is unsat, then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Let  $F$  have  $n + 1$  variables  $p_1, \dots, p_{n+1}$ .

# Resolution

---

- ▶ Inductive hypothesis : If  $F$  has  $\leq n$  variables and is unsat, then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Let  $F$  have  $n + 1$  variables  $p_1, \dots, p_{n+1}$ .

- ▶ Let  $G_0$  be the conjunction of all  $C_i$  in  $F$  such that  $\neg p_{n+1} \notin C_i$ .
- ▶ Let  $G_1$  be the conjunction of all  $C_i$  in  $F$  such that  $p_{n+1} \notin C_i$ .

# Resolution

---

- ▶ Inductive hypothesis : If  $F$  has  $\leq n$  variables and is unsat, then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Let  $F$  have  $n + 1$  variables  $p_1, \dots, p_{n+1}$ .

- ▶ Let  $G_0$  be the conjunction of all  $C_i$  in  $F$  such that  $\neg p_{n+1} \notin C_i$ .
- ▶ Let  $G_1$  be the conjunction of all  $C_i$  in  $F$  such that  $p_{n+1} \notin C_i$ .

- ▶ Clauses in  $F = \text{Clauses in } G_0 \cup \text{Clauses in } G_1$

# Resolution

---

- ▶ Inductive hypothesis : If  $F$  has  $\leq n$  variables and is unsat, then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Let  $F$  have  $n + 1$  variables  $p_1, \dots, p_{n+1}$ .

- ▶ Let  $G_0$  be the conjunction of all  $C_i$  in  $F$  such that  $\neg p_{n+1} \notin C_i$ .
- ▶ Let  $G_1$  be the conjunction of all  $C_i$  in  $F$  such that  $p_{n+1} \notin C_i$ .

- ▶ Clauses in  $F = \text{Clauses in } G_0 \cup \text{Clauses in } G_1$

- ▶ Let  $F_0 = \{C_i - \{p_{n+1}\} \mid C_i \in G_0\}$
- ▶ Let  $F_1 = \{C_i - \{\neg p_{n+1}\} \mid C_i \in G_1\}$

# Resolution

---

Let  $F = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}, \{\neg p_2, \neg p_3\}\}$  and  $n = 2$ .

- ▶  $G_0 = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}\}$ ,  $G_1 = \{\{p_2\}, \{\neg p_2, \neg p_3\}\}$ .
- ▶  $F_0 = \{\{p_1\}, \{p_2\}, \{\neg p_1, \neg p_2\}\}$  and  $F_1 = \{\{p_2\}, \{\neg p_2\}\}$
- ▶ If  $p_{n+1} = \text{false}$  in  $F$ , then  $F$  is equisatisfiable with  $F_0$

# Resolution

---

Let  $F = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}, \{\neg p_2, \neg p_3\}\}$  and  $n = 2$ .

- ▶  $G_0 = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}\}$ ,  $G_1 = \{\{p_2\}, \{\neg p_2, \neg p_3\}\}$ .
- ▶  $F_0 = \{\{p_1\}, \{p_2\}, \{\neg p_1, \neg p_2\}\}$  and  $F_1 = \{\{p_2\}, \{\neg p_2\}\}$
- ▶ If  $p_{n+1} = \text{false}$  in  $F$ , then  $F$  is equisatisfiable with  $F_0$
- ▶ If  $p_{n+1} = \text{true}$  in  $F$ , then  $F$  is equisatisfiable with  $F_1$

# Resolution

---

Let  $F = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}, \{\neg p_2, \neg p_3\}\}$  and  $n = 2$ .

- ▶  $G_0 = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}\}$ ,  $G_1 = \{\{p_2\}, \{\neg p_2, \neg p_3\}\}$ .
- ▶  $F_0 = \{\{p_1\}, \{p_2\}, \{\neg p_1, \neg p_2\}\}$  and  $F_1 = \{\{p_2\}, \{\neg p_2\}\}$
- ▶ If  $p_{n+1} = \text{false}$  in  $F$ , then  $F$  is equisatisfiable with  $F_0$
- ▶ If  $p_{n+1} = \text{true}$  in  $F$ , then  $F$  is equisatisfiable with  $F_1$
- ▶ Hence  $F$  is satisfiable iff  $F_0 \vee F_1$  is.

# Resolution

---

Let  $F = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}, \{\neg p_2, \neg p_3\}\}$  and  $n = 2$ .

- ▶  $G_0 = \{\{p_1, p_3\}, \{p_2\}, \{\neg p_1, \neg p_2, p_3\}\}$ ,  $G_1 = \{\{p_2\}, \{\neg p_2, \neg p_3\}\}$ .
- ▶  $F_0 = \{\{p_1\}, \{p_2\}, \{\neg p_1, \neg p_2\}\}$  and  $F_1 = \{\{p_2\}, \{\neg p_2\}\}$
- ▶ If  $p_{n+1} = \text{false}$  in  $F$ , then  $F$  is equisatisfiable with  $F_0$
- ▶ If  $p_{n+1} = \text{true}$  in  $F$ , then  $F$  is equisatisfiable with  $F_1$
- ▶ Hence  $F$  is satisfiable iff  $F_0 \vee F_1$  is.
- ▶ As  $F$  is unsatisfiable,  $F_0$  and  $F_1$  are both unsatisfiable.

# Resolution

---

- ▶ By induction hypothesis,  $\emptyset \in Res^*(F_0)$  and  $\emptyset \in Res^*(F_1)$ .

# Resolution

---

- ▶ By induction hypothesis,  $\emptyset \in Res^*(F_0)$  and  $\emptyset \in Res^*(F_1)$ .
- ▶ Hence,  $\emptyset \in Res^*(G_0)$  or  $\{p_{n+1}\} \in Res^*(G_0)$ , and  $\emptyset \in Res^*(G_1)$  or  $\{\neg p_{n+1}\} \in Res^*(G_1)$ .

# Resolution

---

- ▶ By induction hypothesis,  $\emptyset \in \text{Res}^*(F_0)$  and  $\emptyset \in \text{Res}^*(F_1)$ .
- ▶ Hence,  $\emptyset \in \text{Res}^*(G_0)$  or  $\{p_{n+1}\} \in \text{Res}^*(G_0)$ , and  $\emptyset \in \text{Res}^*(G_1)$  or  $\{\neg p_{n+1}\} \in \text{Res}^*(G_1)$ .
- ▶ If  $\emptyset \in \text{Res}^*(G_0)$  or  $\emptyset \in \text{Res}^*(G_1)$ , then  $\emptyset \in \text{Res}^*(F)$ .

# Resolution

---

- ▶ By induction hypothesis,  $\emptyset \in \text{Res}^*(F_0)$  and  $\emptyset \in \text{Res}^*(F_1)$ .
- ▶ Hence,  $\emptyset \in \text{Res}^*(G_0)$  or  $\{p_{n+1}\} \in \text{Res}^*(G_0)$ , and  $\emptyset \in \text{Res}^*(G_1)$  or  $\{\neg p_{n+1}\} \in \text{Res}^*(G_1)$ .
- ▶ If  $\emptyset \in \text{Res}^*(G_0)$  or  $\emptyset \in \text{Res}^*(G_1)$ , then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Else,  $\{p_{n+1}\} \in \text{Res}^*(G_0)$  and  $\{\neg p_{n+1}\} \in \text{Res}^*(G_1)$ .

# Resolution

---

- ▶ By induction hypothesis,  $\emptyset \in \text{Res}^*(F_0)$  and  $\emptyset \in \text{Res}^*(F_1)$ .
- ▶ Hence,  $\emptyset \in \text{Res}^*(G_0)$  or  $\{p_{n+1}\} \in \text{Res}^*(G_0)$ , and  $\emptyset \in \text{Res}^*(G_1)$  or  $\{\neg p_{n+1}\} \in \text{Res}^*(G_1)$ .
- ▶ If  $\emptyset \in \text{Res}^*(G_0)$  or  $\emptyset \in \text{Res}^*(G_1)$ , then  $\emptyset \in \text{Res}^*(F)$ .
- ▶ Else,  $\{p_{n+1}\} \in \text{Res}^*(G_0)$  and  $\{\neg p_{n+1}\} \in \text{Res}^*(G_1)$ .
- ▶ Hence  $\emptyset \in \text{Res}^*(F)$ .

# Resolution Summary

---

Given a formula  $\psi$ , convert it into CNF, say  $\zeta$ .  $\psi$  is satisfiable iff  $\emptyset \notin \text{Res}^*(\zeta)$ .

- ▶ If  $\psi$  is unsat, we might get  $\emptyset$  before reaching  $\text{Res}^*(\zeta)$ .
- ▶ If  $\psi$  is sat, then truth tables are faster : stop when some row evaluates to 1.

# CS 228 : Logic in Computer Science

Krishna. S

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...
- ▶ A formula  $\varphi$  is satisfiable iff  $\neg\varphi$  is not valid.

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...
- ▶ A formula  $\varphi$  is satisfiable iff  $\neg\varphi$  is not valid.
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equivalent iff ...

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...
- ▶ A formula  $\varphi$  is satisfiable iff  $\neg\varphi$  is not valid.
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equivalent iff ...
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equisatisfiable iff ...

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...
- ▶ A formula  $\varphi$  is satisfiable iff  $\neg\varphi$  is not valid.
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equivalent iff ...
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equisatisfiable iff ...
- ▶ A disjunction of literals  $L_1 \vee L_2 \vee \dots \vee L_n$  is valid iff ...

# Recap of Basics

---

- ▶ A formula  $\varphi$  is satisfiable when ...
- ▶ A formula  $\varphi$  is valid when ...
- ▶ A formula  $\varphi$  is satisfiable iff  $\neg\varphi$  is not valid.
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equivalent iff ...
- ▶ Two formulae  $\varphi_1$  and  $\varphi_2$  are equisatisfiable iff ...
- ▶ A disjunction of literals  $L_1 \vee L_2 \vee \dots \vee L_n$  is valid iff ...
- ▶ A conjunction of literals  $L_1 \wedge L_2 \wedge \dots \wedge L_n$  is satisfiable iff ...

# Normal Forms : CNF Validity

---

Let  $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_n$  be in CNF.

- ▶ Checking if  $\varphi$  is satisfiable is NP-complete.
- ▶ Checking if  $\varphi$  is valid is polynomial time. Why?
- ▶ Question raised in class : If validity is polytime, so should be satisfiability. Is this true?

# CS 228 : Logic in Computer Science

Krishna. S

# Normal Forms : CNF Validity

---

Let  $\varphi = C_1 \wedge C_2 \wedge \dots \wedge C_n$  be in CNF.

- ▶ Checking if  $\varphi$  is satisfiable is NP-complete.
- ▶ Checking if  $\varphi$  is valid is polynomial time. Why?
- ▶ Question raised in class : If validity check is polynomial time, so should be satisfiability. Is this true?
- ▶ If  $\varphi$  is valid, it is indeed satisfiable
- ▶ If  $\varphi$  is not valid, then...?

# Normal Forms : DNF Satisfiability

---

Let  $\varphi = D_1 \vee D_2 \vee \dots \vee D_n$  be in DNF.

- ▶ Checking if  $\varphi$  is valid is NP-complete. Why?
- ▶ Checking if  $\varphi$  is satisfiable is polynomial time. Why?

# Normal Forms from Truth Tables

---

Assume you are given the truth table of a formula  $\varphi$ . Then it is very easy to obtain the equivalent CNF/DNF of  $\varphi$ .

# Normal Forms from Truth Tables

---

Assume you are given the truth table of a formula  $\varphi$ . Then it is very easy to obtain the equivalent CNF/DNF of  $\varphi$ .

- ▶ Consider for example  $\varphi = p \leftrightarrow q$ .
- ▶ Truth table of  $\varphi$ :  $\varphi$  is false when  $p = T, q = F$  and  $p = F, q = T$ .

# Normal Forms from Truth Tables

---

Assume you are given the truth table of a formula  $\varphi$ . Then it is very easy to obtain the equivalent CNF/DNF of  $\varphi$ .

- ▶ Consider for example  $\varphi = p \leftrightarrow q$ .
- ▶ Truth table of  $\varphi$ :  $\varphi$  is false when  $p = T, q = F$  and  $p = F, q = T$ .
- ▶ CNF equivalent is  $(\neg p \vee q) \wedge (p \vee \neg q)$ .

# CNF to DNF Sizes

---

- ▶  $\varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots (p_n \vee q_n)$
- ▶ What is the equivalent DNF formula?

# CNF to DNF Sizes

---

►  $\varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots (p_n \vee q_n)$

► What is the equivalent DNF formula?

►

$$\varphi' = \bigvee_{S \subseteq \{1, \dots, n\}} \left( \bigwedge_{i \in S} p_i \wedge \bigwedge_{i \notin S} q_i \right)$$

# CNF to DNF Sizes

---

$$\triangleright \varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots (p_n \vee q_n)$$

$\triangleright$  What is the equivalent DNF formula?

$\triangleright$

$$\varphi' = \bigvee_{S \subseteq \{1, \dots, n\}} \left( \bigwedge_{i \in S} p_i \wedge \bigwedge_{i \notin S} q_i \right)$$

$\triangleright$  Prove that any equivalent DNF formula has  $2^n$  clauses

# CNF to DNF Sizes

---

$$\varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots \wedge (p_n \vee q_n)$$

- ▶ What is the equivalent DNF formula?

- ▶

$$\varphi' = \bigvee_{S \subseteq \{1, \dots, n\}} \left( \bigwedge_{i \in S} p_i \wedge \bigwedge_{i \notin S} q_i \right)$$

- ▶ Prove that any equivalent DNF formula has  $2^n$  clauses
- ▶ Call an assignment *minimal* if it maps exactly one of  $p_i, q_i$  to 1

# CNF to DNF Sizes

---

►  $\varphi = (p_1 \vee q_1) \wedge (p_2 \vee q_2) \wedge \dots (p_n \vee q_n)$

► What is the equivalent DNF formula?

►

$$\varphi' = \bigvee_{S \subseteq \{1, \dots, n\}} \left( \bigwedge_{i \in S} p_i \wedge \bigwedge_{i \notin S} q_i \right)$$

► Prove that any equivalent DNF formula has  $2^n$  clauses

► Call an assignment *minimal* if it maps exactly one of  $p_i, q_i$  to 1

► There are  $2^n$  *minimal* assignments, satisfying clauses in  $\varphi'$

► Show that no two *minimal* assignments satisfy the same clause of  $\varphi'$  (hence there must be  $2^n$  clauses in  $\varphi'$ )

# CNF to DNF Sizes

---

- ▶ Let  $\alpha$  and  $\beta$  be two minimal assignments such that  $\alpha(p_i) \neq \beta(p_i)$  for  $i \in \{1, \dots, n\}$

# CNF to DNF Sizes

---

- ▶ Let  $\alpha$  and  $\beta$  be two minimal assignments such that  $\alpha(p_i) \neq \beta(p_i)$  for  $i \in \{1, \dots, n\}$
- ▶ Define a new assignment  $\min(\alpha, \beta)$  as a pointwise min of  $\alpha, \beta$
- ▶  $\min(\alpha, \beta)(p) = \min(\alpha(p), \beta(p))$  for each variable  $p$  with the assumption that  $0 < 1$ , 0 represents false and 1 represents true

# CNF to DNF Sizes

---

- ▶ Let  $\alpha$  and  $\beta$  be two minimal assignments such that  $\alpha(p_i) \neq \beta(p_i)$  for  $i \in \{1, \dots, n\}$
- ▶ Define a new assignment  $\min(\alpha, \beta)$  as a pointwise min of  $\alpha, \beta$
- ▶  $\min(\alpha, \beta)(p) = \min(\alpha(p), \beta(p))$  for each variable  $p$  with the assumption that  $0 < 1$ , 0 represents false and 1 represents true
- ▶  $\min(\alpha, \beta) \not\models p_i \vee q_i$ ,  $\min(\alpha, \beta) \not\models \varphi'$

# CNF to DNF Sizes

---

- ▶ Let  $\alpha$  and  $\beta$  be two minimal assignments such that  $\alpha(p_i) \neq \beta(p_i)$  for  $i \in \{1, \dots, n\}$
- ▶ Define a new assignment  $\min(\alpha, \beta)$  as a pointwise min of  $\alpha, \beta$
- ▶  $\min(\alpha, \beta)(p) = \min(\alpha(p), \beta(p))$  for each variable  $p$  with the assumption that  $0 < 1$ , 0 represents false and 1 represents true
- ▶  $\min(\alpha, \beta) \not\models p_i \vee q_i$ ,  $\min(\alpha, \beta) \not\models \varphi'$
- ▶ However, if  $\alpha \models D_j$  and  $\beta \models D_j$  for some clause  $D_j$  of  $\varphi'$ , then  $\min(\alpha, \beta) \models D_j$  and hence  $\min(\alpha, \beta) \models \varphi'$ , a contradiction.

Think of an example where DNF to CNF explodes.

# CS 228 : Logic in Computer Science

Krishna. S

# CNF Explosion

---

Consider the formula  $\varphi = (p_1 \wedge p_2 \dots \wedge p_n) \vee (q_1 \wedge q_2 \dots \wedge q_m)$

- ▶ What is the equivalent CNF formula?
- ▶  $\bigwedge_{i \in \{1, \dots, n\}, j \in \{1, \dots, m\}} (p_i \vee q_j)$  has  $mn$  clauses
- ▶ Distributivity explodes the formula

# Tseitin Encoding : The Idea

---

- ▶ Introducing fresh variables, Tseitin encoding can give an equisatisfiable formula without exponential explosion.
- ▶  $\varphi = p \vee (q \wedge r)$
- ▶ Replace  $q \wedge r$  with a fresh variable  $x$  and add a clause which asserts that  $x$  simulates  $q \wedge r$
- ▶  $(p \vee x) \wedge (x \leftrightarrow (q \wedge r))$
- ▶ It is enough to consider (Why?)  $(p \vee x) \wedge (x \rightarrow (q \wedge r))$  which is  $(p \vee x) \wedge (\neg x \vee q) \wedge (\neg x \vee r)$

# Tseitin Encoding

---

- ▶ Assume the input formula is in NNF (all negations attached only to literals) and has only  $\wedge, \vee$
- ▶ Replace each  $G_1 \wedge \cdots \wedge G_n$  just below a  $\vee$  with a fresh variable  $p$  and conjunct  $(\neg p \vee G_1) \wedge \cdots \wedge (\neg p \vee G_n)$  (same as  $p \rightarrow G_1 \wedge \cdots \wedge G_n$ ).

# Tseitin Encoding

---

- ▶  $\varphi = (p_1 \wedge p_2 \cdots \wedge p_n) \vee (q_1 \wedge q_2 \cdots \wedge q_m)$
- ▶ Choose fresh variables  $x, y$
- ▶  $\psi = (x \vee y) \wedge \bigwedge_{i \in \{1, \dots, n\}} (\neg x \vee p_i) \wedge \bigwedge_{j \in \{1, \dots, m\}} (\neg y \vee q_j)$  has  $m + n + 1$  clauses
- ▶  $\varphi$  and  $\psi$  are equisatisfiable. Prove.

(DPLL)Davis-Putnam-Loveland-Logemann Method

# DPLL

---

- ▶ DPLL combines search and deduction to decide CNF satisfiability
- ▶ Underlies most modern SAT solvers

# Partial Assignments

---

An assignment is a function  $\alpha : V \rightarrow \{0, 1\}$  which maps each variable to true(1) or false (0). A partial assignment  $\alpha$  is one under which some variables are unassigned.

# Partial Assignments

---

An assignment is a function  $\alpha : V \rightarrow \{0, 1\}$  which maps each variable to true(1) or false (0). A partial assignment  $\alpha$  is one under which some variables are unassigned.

- ▶ Under a partial assignment  $\alpha$ , the **state** of a variable  $v$  is true if  $\alpha(v) = 1$ , false if  $\alpha(v) = 0$ , and unassigned otherwise.
- ▶ Let  $V = \{x, y, z\}$  and let  $\alpha(x) = 1, \alpha(y) = 0$ . Then the state of  $x$  under  $\alpha$  is true, state of  $y$  is false, and the state of  $z$  is unassigned.

# State of a Clause

---

Assume we have a formula in CNF. Under a partial assignment  $\alpha$ ,

- ▶ a clause  $C$  is true if there exists some literal  $\ell$  in  $C$  whose state is true
- ▶ a clause  $C$  is false if the state of all literals in  $C$  is false
- ▶ Otherwise, the state of  $C$  is unassigned

# State of a Clause

---

Assume we have a formula in CNF. Under a partial assignment  $\alpha$ ,

- ▶ a clause  $C$  is true if there exists some literal  $\ell$  in  $C$  whose state is true
- ▶ a clause  $C$  is false if the state of all literals in  $C$  is false
- ▶ Otherwise, the state of  $C$  is unassigned

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

# State of a Clause

---

Assume we have a formula in CNF. Under a partial assignment  $\alpha$ ,

- ▶ a clause  $C$  is true if there exists some literal  $\ell$  in  $C$  whose state is true
- ▶ a clause  $C$  is false if the state of all literals in  $C$  is false
- ▶ Otherwise, the state of  $C$  is unassigned

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

- ▶ The state of  $C = x \vee y \vee z$  is true

# State of a Clause

---

Assume we have a formula in CNF. Under a partial assignment  $\alpha$ ,

- ▶ a clause  $C$  is true if there exists some literal  $\ell$  in  $C$  whose state is true
- ▶ a clause  $C$  is false if the state of all literals in  $C$  is false
- ▶ Otherwise, the state of  $C$  is unassigned

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

- ▶ The state of  $C = x \vee y \vee z$  is true
- ▶ The state of  $C = x \vee \neg y \vee z$  is unassigned

# State of a Clause

---

Assume we have a formula in CNF. Under a partial assignment  $\alpha$ ,

- ▶ a clause  $C$  is true if there exists some literal  $\ell$  in  $C$  whose state is true
- ▶ a clause  $C$  is false if the state of all literals in  $C$  is false
- ▶ Otherwise, the state of  $C$  is unassigned

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

- ▶ The state of  $C = x \vee y \vee z$  is true
- ▶ The state of  $C = x \vee \neg y \vee z$  is unassigned
- ▶ The state of  $C = x \vee \neg y$  is false

# State of a Formula

---

Under a partial assignment  $\alpha$ ,

- ▶ A CNF formula  $F$  is true if for each  $C \in F$ ,  $C$  is true
- ▶ A CNF formula  $F$  is false if there  $C \in F$  such that  $C$  is false
- ▶ Otherwise,  $F$  is unassigned.

# State of a Formula

---

Under a partial assignment  $\alpha$ ,

- ▶ A CNF formula  $F$  is true if for each  $C \in F$ ,  $C$  is true
- ▶ A CNF formula  $F$  is false if there  $C \in F$  such that  $C$  is false
- ▶ Otherwise,  $F$  is unassigned.

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

# State of a Formula

---

Under a partial assignment  $\alpha$ ,

- ▶ A CNF formula  $F$  is true if for each  $C \in F$ ,  $C$  is true
- ▶ A CNF formula  $F$  is false if there  $C \in F$  such that  $C$  is false
- ▶ Otherwise,  $F$  is unassigned.

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

- ▶ The state of  $F = (x \vee y \vee z) \wedge (x \vee \neg y \vee z)$  is unassigned

# State of a Formula

---

Under a partial assignment  $\alpha$ ,

- ▶ A CNF formula  $F$  is true if for each  $C \in F$ ,  $C$  is true
- ▶ A CNF formula  $F$  is false if there  $C \in F$  such that  $C$  is false
- ▶ Otherwise,  $F$  is unassigned.

Consider the partial assignment  $\alpha(x) = 0, \alpha(y) = 1$ .

- ▶ The state of  $F = (x \vee y \vee z) \wedge (x \vee \neg y \vee z)$  is unassigned
- ▶ The state of  $F = (x \vee y \vee z) \wedge (x \vee \neg y)$  is false

# Unit Clause and Unit Literal

---

Let  $C$  be a clause and  $\alpha$  a partial assignment. Then

- ▶  $C$  is a unit clause under  $\alpha$  if there is a literal  $\ell \in C$  which is unassigned, and the rest are false.
- ▶ Then  $\ell$  is a unit literal under  $\alpha$ .

# Unit Clause and Unit Literal

---

Let  $C$  be a clause and  $\alpha$  a partial assignment. Then

- ▶  $C$  is a unit clause under  $\alpha$  if there is a literal  $\ell \in C$  which is unassigned, and the rest are false.
- ▶ Then  $\ell$  is a unit literal under  $\alpha$ .

Let  $\alpha(x) = 0, \alpha(y) = 1$  be a partial assignment.

- ▶  $C = x \vee \neg y \vee \neg z$  is a unit clause and  $\neg z$  is a unit literal

# Unit Clause and Unit Literal

---

Let  $C$  be a clause and  $\alpha$  a partial assignment. Then

- ▶  $C$  is a unit clause under  $\alpha$  if there is a literal  $\ell \in C$  which is unassigned, and the rest are false.
- ▶ Then  $\ell$  is a unit literal under  $\alpha$ .

Let  $\alpha(x) = 0, \alpha(y) = 1$  be a partial assignment.

- ▶  $C = x \vee \neg y \vee \neg z$  is a unit clause and  $\neg z$  is a unit literal
- ▶  $C = x \vee \neg y \vee \neg z \vee w$  is not a unit clause

# Unit Clause and Unit Literal

---

Let  $C$  be a clause and  $\alpha$  a partial assignment. Then

- ▶  $C$  is a unit clause under  $\alpha$  if there is a literal  $\ell \in C$  which is unassigned, and the rest are false.
- ▶ Then  $\ell$  is a unit literal under  $\alpha$ .

Let  $\alpha(x) = 0, \alpha(y) = 1$  be a partial assignment.

- ▶  $C = x \vee \neg y \vee \neg z$  is a unit clause and  $\neg z$  is a unit literal
- ▶  $C = x \vee \neg y \vee \neg z \vee w$  is not a unit clause
- ▶  $C = x \vee \neg y$  is not a unit clause

# DPLL

---

DPLL maintains a partial assignment, to begin with the empty assignment.

- ▶ Assigns unassigned variables 0 or 1 randomly
- ▶ Sometimes, forced to assign 0 or 1 to unit literals

# DPLL Actions

---

- ▶ DPLL has 3 actions : decisions, unit propagation and backtracking
- ▶ Decisions : Decide an assignment for a variable (random choice)
- ▶ Implied assignments or unit propagation : to deal with unit literals
- ▶ Backtrack when in a conflict

# DPLL Algorithm

---

- ▶ At any time, the state of the algorithm is a pair  $(F, \alpha)$  where  $F$  is the CNF and  $\alpha$  is a partial assignment
- ▶ A state  $(F, \alpha)$  is **successful** if  $\alpha$  sets some literal in each clause of  $F$  to be true
- ▶ A **conflict** state is one where  $\alpha$  sets all literals in some clause of  $F$  to be false

# DPLL Algorithm

---

- ▶ Let  $F|\alpha$  denote the set of clauses obtained by deleting from  $F$ , any clause containing a true literal from  $\alpha$ , and deleting from each remaining clause, all literals false under  $\alpha$ . Let  $\alpha(x) = 0, \alpha(y) = 1$ .
- ▶ For  $F = (x \vee y \vee z) \wedge (x \vee \neg y \vee \neg z)$ ,  $F|\alpha = \{\neg z\}$
- ▶ For  $F = (x \vee y) \wedge (\neg x \vee \neg y)$ ,  $F|\alpha = \{\}$ .
- ▶ For  $F = (x \vee \neg y)$ ,  $\perp \in F|\alpha$
- ▶ If  $(F, \alpha)$  is successful, then  $F|\alpha = \{\}$
- ▶ If  $(F, \alpha)$  is in conflict, then the empty clause  $\perp$  is in  $F|\alpha$ .

# The DPLL Algorithm

---

Input : CNF formula  $F$ .

1. Initialise  $\alpha$  as the empty assignment
2. While there is a unit clause  $L$  in  $F|\alpha$ , add  $L = 1$  to  $\alpha$  (unit propagation)
3. If  $F|\alpha$  contains no clauses, then stop and output  $\alpha$
4. If  $F|\alpha$  contains the empty clause, then apply the learning procedure to add a new clause  $C$  to  $F$ . If it is the empty clause, output UNSAT. Otherwise, backtrack to the highest level at which  $C$  is a unit clause, go to line 2.
5. Decide on a new assignment  $p = b$  to be added to  $\alpha$ , goto line 2.

# DPLL Example

$$c_1 = \neg p_1 \vee p_2$$

$$c_2 = \neg p_1 \vee p_3 \vee p_5$$

$$c_3 = \neg p_2 \vee p_4$$

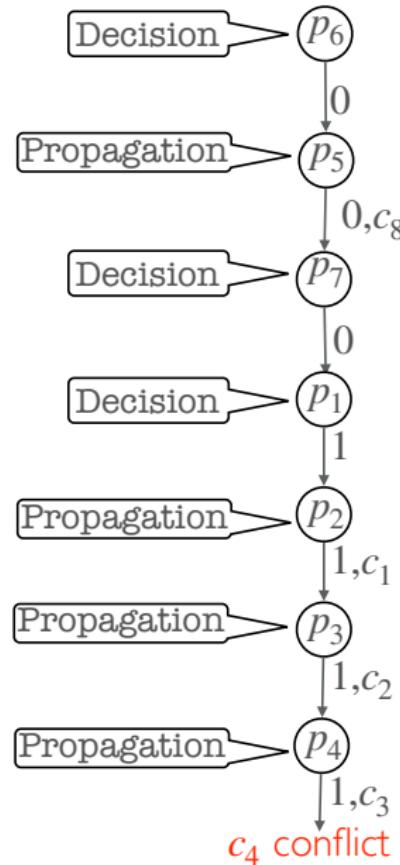
$$c_4 = \neg p_3 \vee \neg p_4$$

$$c_5 = p_1 \vee p_5 \vee \neg p_2$$

$$c_6 = p_2 \vee p_3$$

$$c_7 = p_2 \vee \neg p_3 \vee p_7$$

$$c_8 = p_6 \vee \neg p_5$$



## Clause Learning

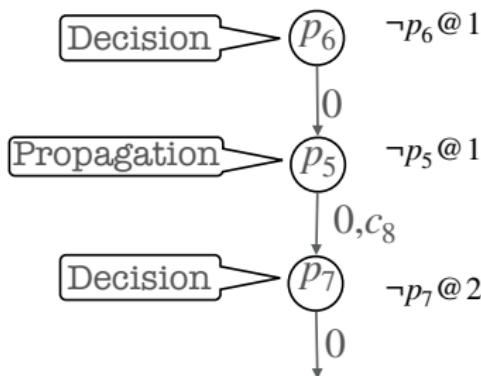
# Run of DPLL

---

The partial assignment in construction is called a **a run of DPLL**. In the previous slide, the run ended in a conflict.

# Decision Level

During a run, the decision level of a true literal is the number of decisions after which the literal was made true.



# Implication Graphs

---

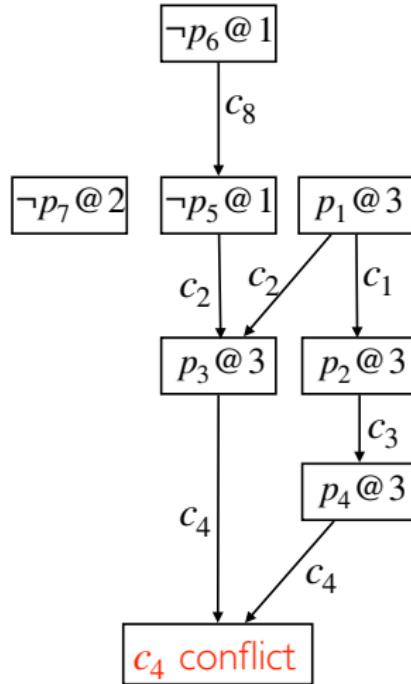
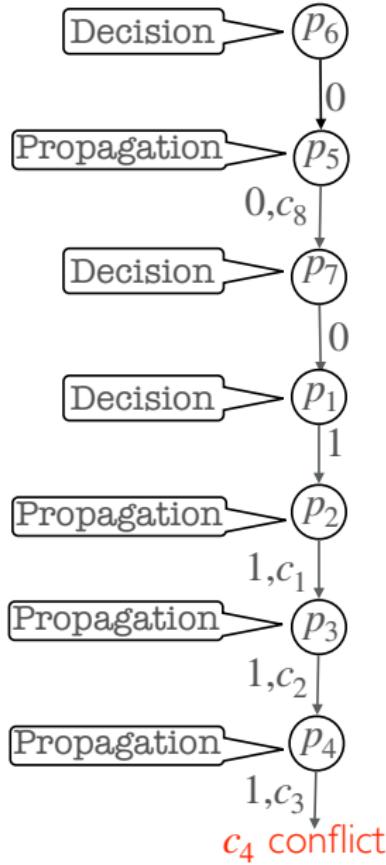
During a DPLL run, we maintain a data structure called an implication graph.

Under a partial assignment  $\alpha$ , the implication graph  $G = (V, E)$ ,

- ▶  $V$  is the set of true literals under  $\alpha$ , and the conflict node
- ▶  $E = \{(\ell_1, \ell_2) \mid \neg\ell_1 \text{ belongs to the clause due to which unit propagation made } \ell_2 \text{ true}\}$

Each node is annotated with the decision level.

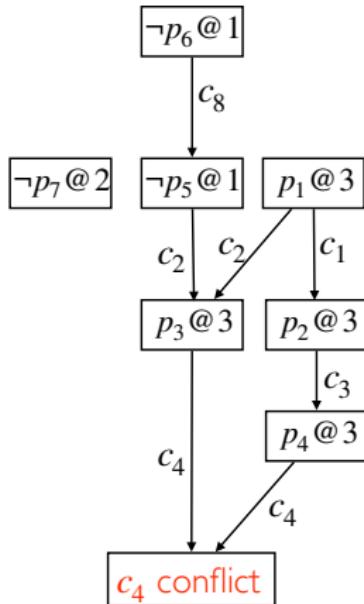
$$\begin{aligned}
 c_1 &= \neg p_1 \vee p_2 \\
 c_2 &= \neg p_1 \vee p_3 \vee p_5 \\
 c_3 &= \neg p_2 \vee p_4 \\
 c_4 &= \neg p_3 \vee \neg p_4 \\
 c_5 &= p_1 \vee p_5 \vee \neg p_2 \\
 c_6 &= p_2 \vee p_3 \\
 c_7 &= p_2 \vee \neg p_3 \vee p_7 \\
 c_8 &= p_6 \vee \neg p_5
 \end{aligned}$$



# Conflict Clause

Traverse the implication graph backwards to find the set of decisions that created a conflict. The negations of the causing decisions is the **conflict clause**.

$$\begin{aligned}c_1 &= \neg p_1 \vee p_2 \\c_2 &= \neg p_1 \vee p_3 \vee p_5 \\c_3 &= \neg p_2 \vee p_4 \\c_4 &= \neg p_3 \vee \neg p_4 \\c_5 &= p_1 \vee p_5 \vee \neg p_2 \\c_6 &= p_2 \vee p_3 \\c_7 &= p_2 \vee \neg p_3 \vee p_7 \\c_8 &= p_6 \vee \neg p_5\end{aligned}$$



Lecture 09, DPLL  
Conflict clause:  $p_6 \vee \neg p_1$  is added : resolve  $c_4$  with  $c_3, c_1, c_2, c_8$

# Clause Learning

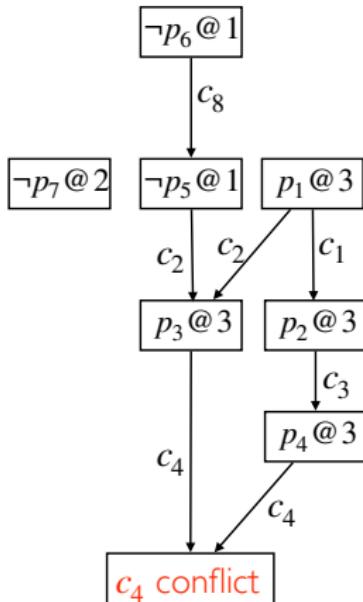
---

- ▶ We add the conflict clause to the input set of clauses
- ▶ backtrack to the second last conflicting decision, and proceed like DPLL

## Adding the conflict clause

- ▶ does not affect satisfiability of the original formula (think of resolution)
- ▶ ensures that the conflicting partial assignment will not be tried again

$$\begin{aligned}
 c_1 &= \neg p_1 \vee p_2 \\
 c_2 &= \neg p_1 \vee p_3 \vee p_5 \\
 c_3 &= \neg p_2 \vee p_4 \\
 c_4 &= \neg p_3 \vee \neg p_4 \\
 c_5 &= p_1 \vee p_5 \vee \neg p_2 \\
 c_6 &= p_2 \vee p_3 \\
 c_7 &= p_2 \vee \neg p_3 \vee p_7 \\
 c_8 &= p_6 \vee \neg p_5
 \end{aligned}$$



The second last decision is  $p_6 = 0$ . Unit propagation will force  $p_1 = 0$ .

The combination  $p_6 = 0, p_1 = 1$  will not be tried again.

$$c_1 = \neg p_1 \vee p_2$$

$$c_2 = \neg p_1 \vee p_3 \vee p_5$$

$$c_3 = \neg p_2 \vee p_4$$

$$c_4 = \neg p_3 \vee \neg p_4$$

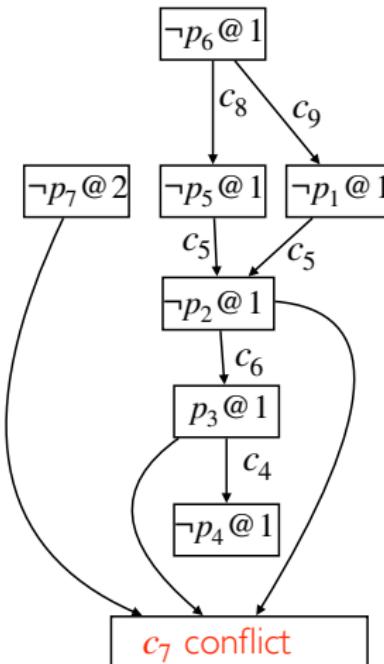
$$c_5 = p_1 \vee p_5 \vee \neg p_2$$

$$c_6 = p_2 \vee p_3$$

$$c_7 = p_2 \vee \neg p_3 \vee p_7$$

$$c_8 = p_6 \vee \neg p_5$$

$$c_9 = p_6 \vee \neg p_1$$



Conflict clause :  $p_7 \vee p_6$  is added and backtrack

Set  $p_7 = 1$  by unit propagation.

$$c_1 = \neg p_1 \vee p_2$$

$$c_2 = \neg p_1 \vee p_3 \vee p_5$$

$$c_3 = \neg p_2 \vee p_4$$

$$c_4 = \neg p_3 \vee \neg p_4$$

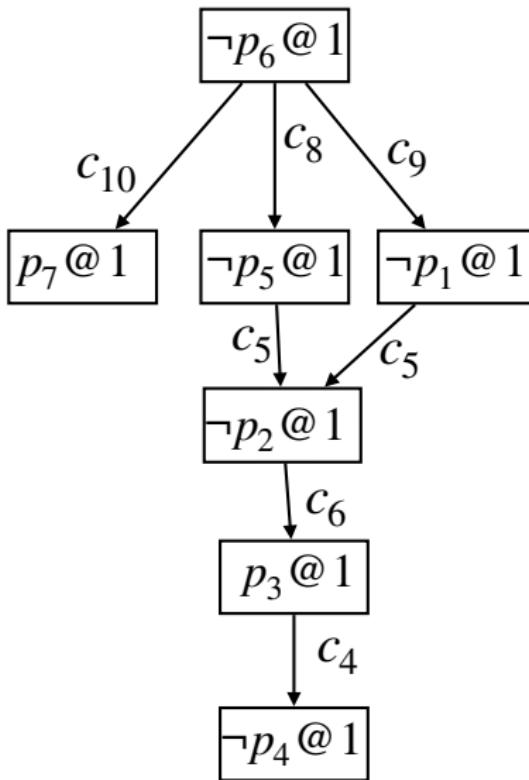
$$c_5 = p_1 \vee p_5 \vee \neg p_2$$

$$c_6 = p_2 \vee p_3$$

$$c_7 = p_2 \vee \neg p_3 \vee p_7$$

$$c_8 = p_6 \vee \neg p_5$$

$$c_9 = p_6 \vee \neg p_1$$



Lecture 09 DPLL

424

# CS 228 : Logic in Computer Science

Krishna. S

# The DPLL Algorithm

---

Input : CNF formula  $F$ .

1. Initialise  $\alpha$  as the empty assignment
2. While there is a unit clause  $L$  in  $F|\alpha$ , add  $L = 1$  to  $\alpha$  (unit propagation)
3. If  $F|\alpha$  contains no clauses, then stop and output  $\alpha$
4. If  $F|\alpha$  contains the empty clause, then apply the learning procedure to add a new clause  $C$  to  $F$ . If it is the empty clause, output UNSAT. Otherwise, backtrack to the highest level at which  $C$  is a unit clause, go to line 2.
5. Decide on a new assignment  $p = b$  to be added to  $\alpha$ , goto line 2.

# DPLL Example

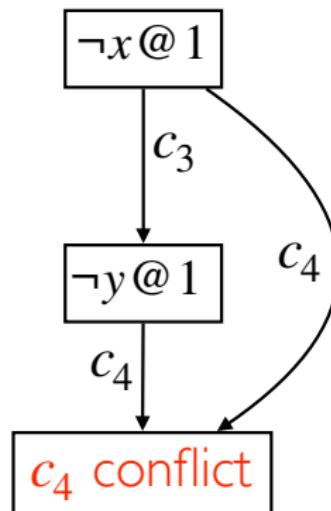
---

$$c_1 = \neg x \vee \neg y$$

$$c_2 = \neg x \vee y$$

$$c_3 = x \vee \neg y$$

$$c_4 = x \vee y$$



Clause learnt :  $x$  (Resolve  $c_4$  with  $c_3$ )

# DPLL Example

---

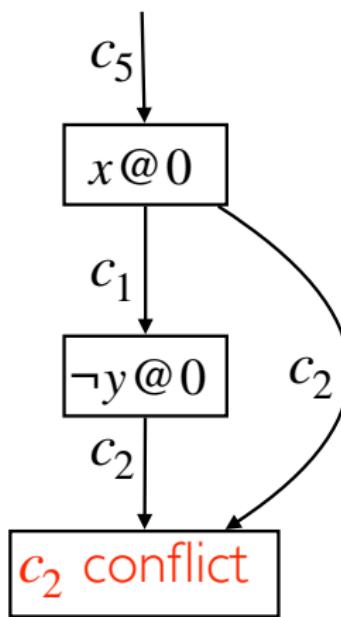
$$c_1 = \neg x \vee \neg y$$

$$c_2 = \neg x \vee y$$

$$c_3 = x \vee \neg y$$

$$c_4 = x \vee y$$

$$c_5 = x$$



Clause learnt : Resolve  $c_2$  with  $c_1, c_5$ . Empty clause.

# DPLL Correctness

---

## Termination

A sequence of decisions which lead to a conflict cannot be repeated : the variables in the learned clause are all decision variables. In a future assignment, if all but one of these are set to false, the remaining one will not be a decision variable.

# DPLL Correctness

---

## Termination

A sequence of decisions which lead to a conflict cannot be repeated : the variables in the learned clause are all decision variables. In a future assignment, if all but one of these are set to false, the remaining one will not be a decision variable.

## Correctness

Correctness is straightforward :  $F \vdash$  the learned clause. Thus, if the empty clause is learnt, then  $F$  is unsat. Otherwise, if DPLL terminates with a satisfying assignment  $\alpha$ , then the input formula is also satisfied by  $\alpha$ .

# Modern SAT Solvers

---

Numerous enhancements/heuristics

- ▶ Decision heuristics to choose decision variables
- ▶ Random restarts

## First Order Logic

Extends propositional logic

- ▶ Propositional logic : atomic formulas have no internal structure
- ▶ FOL : atomic formulas are predicates that assert a relationship between certain elements
- ▶ Quantification in FOL : ability to assert that a certain property holds for all elements or only for some element.
- ▶ Formulae in FOL are over some signature.

# Signatures

---

- ▶ A **vocabulary** or **signature**  $\tau$  is a set consisting of
  - ▶ constants  $c_1, c_2, \dots$
  - ▶ Relation symbols  $R_1, R_2, \dots$ , each with some arity  $k$ , denoted  $R_i^k$
  - ▶ Function symbols  $f_1, \dots$  each with some arity  $k$ , denoted  $f_i^k$
- ▶ We look at finite signatures
- ▶  $\tau = (E^2, F^3, f^1)$  is a finite signature with two relations,  $E$  with arity 2 and  $F$  with arity 3, and a function  $f$  with arity 1

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**
- ▶ Constants, relations and functions from  $\tau$

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**
- ▶ Constants, relations and functions from  $\tau$
- ▶ The symbols  $\rightarrow, \neg, \wedge, \vee$

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**
- ▶ Constants, relations and functions from  $\tau$
- ▶ The symbols  $\rightarrow, \neg, \wedge, \vee$
- ▶ The symbol  $\forall$  called the **universal quantifier**

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**
- ▶ Constants, relations and functions from  $\tau$
- ▶ The symbols  $\rightarrow, \neg, \wedge, \vee$
- ▶ The symbol  $\forall$  called the **universal quantifier**
- ▶ The symbol  $\exists$  called the **existential quantifier**

# Symbols in First Order Logic

---

Formulae of FO, over signature  $\tau$ , are sequences of symbols, where each symbol is one of the following:

- ▶ The symbols  $\perp, \top$  called **false, true**
- ▶ An element of the infinite set  $\mathcal{V} = \{x_1, x_2, \dots\}$  called **variables**
- ▶ Constants, relations and functions from  $\tau$
- ▶ The symbols  $\rightarrow, \neg, \wedge, \vee$
- ▶ The symbol  $\forall$  called the **universal quantifier**
- ▶ The symbol  $\exists$  called the **existential quantifier**
- ▶ The symbols ( and ) called **paranthesis**

# Terms

---

Given a signature  $\tau$ , the set of  $\tau$ -terms are defined inductively as follows.

- ▶ Each variable is a term
- ▶ Each constant symbol is a term
- ▶ If  $t_1, \dots, t_k$  are terms and  $f$  is a  $k$ -ary function, then  $f(t_1, \dots, t_k)$  is a term
- ▶ Ground Terms : Terms without variables. For instance  $f(c_1, \dots, c_k)$  for constants  $c_1, \dots, c_k$ .

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff
- ▶ If  $t_1, t_2$  are either variables or constants in  $\tau$ , then  $t_1 = t_2$  is a wff

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff
- ▶ If  $t_1, t_2$  are either variables or constants in  $\tau$ , then  $t_1 = t_2$  is a wff
- ▶ If  $t_i$  is a term, for  $1 \leq i \leq k$  and  $R$  is a  $k$ -ary relation symbol in  $\tau$ , then  $R(t_1, \dots, t_k)$  is a wff

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff
- ▶ If  $t_1, t_2$  are either variables or constants in  $\tau$ , then  $t_1 = t_2$  is a wff
- ▶ If  $t_i$  is a term, for  $1 \leq i \leq k$  and  $R$  is a  $k$ -ary relation symbol in  $\tau$ , then  $R(t_1, \dots, t_k)$  is a wff
- ▶ If  $\varphi$  and  $\psi$  are wff, then  $\varphi \rightarrow \psi, \varphi \wedge \psi, \varphi \vee \psi, \neg \psi$  are all wff

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff
- ▶ If  $t_1, t_2$  are either variables or constants in  $\tau$ , then  $t_1 = t_2$  is a wff
- ▶ If  $t_i$  is a term, for  $1 \leq i \leq k$  and  $R$  is a  $k$ -ary relation symbol in  $\tau$ , then  $R(t_1, \dots, t_k)$  is a wff
- ▶ If  $\varphi$  and  $\psi$  are wff, then  $\varphi \rightarrow \psi, \varphi \wedge \psi, \varphi \vee \psi, \neg \psi$  are all wff
- ▶ If  $\varphi$  is a wff and  $x$  is a variable, then  $(\forall x)\varphi$  and  $(\exists x)\varphi$  are wff

# Well formed Formulae

---

A well-formed formula (wff) over a signature  $\tau$  is inductively defined as follows:

- ▶  $\perp, \top$  are wff
- ▶ If  $t_1, t_2$  are either variables or constants in  $\tau$ , then  $t_1 = t_2$  is a wff
- ▶ If  $t_i$  is a term, for  $1 \leq i \leq k$  and  $R$  is a  $k$ -ary relation symbol in  $\tau$ , then  $R(t_1, \dots, t_k)$  is a wff
- ▶ If  $\varphi$  and  $\psi$  are wff, then  $\varphi \rightarrow \psi, \varphi \wedge \psi, \varphi \vee \psi, \neg\psi$  are all wff
- ▶ If  $\varphi$  is a wff and  $x$  is a variable, then  $(\forall x)\varphi$  and  $(\exists x)\varphi$  are wff
- ▶ The second and third are **atomic** formulae.
- ▶ If a formula  $F$  occurs as part of another formula  $G$ , then  $F$  is called a **sub formula** of  $G$ .

# Logical Abbreviations : Boolean Connectives

---

- ▶  $\neg\varphi = \varphi \rightarrow \perp$
- ▶  $\top = \neg\perp$
- ▶  $\varphi \vee \psi = \neg\varphi \rightarrow \psi$
- ▶  $\varphi \wedge \psi = \neg(\neg\varphi \vee \neg\psi)$
- ▶  $\exists x.\varphi = \neg(\forall x.\neg\varphi)$
- ▶ Precedence of operators : Quantifiers and negation highest, followed by  $\vee, \wedge$ , followed by  $\rightarrow$ .
  - ▶  $\forall x P(x) \wedge R(x)$  is  $[\forall x.[P(x)]] \wedge R(x)$

# An Example

---

Consider the signature  $\tau = \{R\}$  where  $R$  is a binary relation. The following are FO formulae over this signature.

- ▶  $\forall x R(x, x)$  Reflexivity
- ▶  $\forall x (R(x, x) \rightarrow \perp)$  Irreflexivity
- ▶  $\forall x \forall y (R(x, y) \rightarrow R(y, x))$  Symmetry
- ▶  $\forall x \forall y \forall z (R(x, y) \rightarrow (R(y, z) \rightarrow R(x, z)))$  Transitivity

# CS 228 : Logic in Computer Science

Krishna. S

## First-Order Logic : Semantics

# Structures

---

- ▶ A structure  $\mathcal{A}$  of signature  $\tau$  consists of

# Structures

---

- ▶ A structure  $\mathcal{A}$  of signature  $\tau$  consists of
  - ▶ A non-empty set  $A$  or  $u(\mathcal{A})$  called the **universe**

# Structures

---

- ▶ A structure  $\mathcal{A}$  of signature  $\tau$  consists of
  - ▶ A non-empty set  $A$  or  $u(\mathcal{A})$  called the **universe**
  - ▶ For each constant  $c$  in the signature  $\tau$ , a fixed element  $c_{\mathcal{A}}$  is assigned from the universe  $A$

# Structures

---

- ▶ A structure  $\mathcal{A}$  of signature  $\tau$  consists of
  - ▶ A non-empty set  $A$  or  $u(\mathcal{A})$  called the **universe**
  - ▶ For each constant  $c$  in the signature  $\tau$ , a fixed element  $c_{\mathcal{A}}$  is assigned from the universe  $A$
  - ▶ For each  $k$ -ary relation  $R^k$  in the signature  $\tau$ , a set of  $k$ -tuples from  $A^k$  is assigned to  $R^{\mathcal{A}}$

# Structures

---

- ▶ A structure  $\mathcal{A}$  of signature  $\tau$  consists of
  - ▶ A non-empty set  $A$  or  $u(\mathcal{A})$  called the **universe**
  - ▶ For each constant  $c$  in the signature  $\tau$ , a fixed element  $c_{\mathcal{A}}$  is assigned from the universe  $A$
  - ▶ For each  $k$ -ary relation  $R^k$  in the signature  $\tau$ , a set of  $k$ -tuples from  $A^k$  is assigned to  $R^{\mathcal{A}}$
  - ▶ The structure  $\mathcal{A}$  is finite if  $A$  (or  $u(\mathcal{A})$ ) is finite

# Examples of Structures : A Graph

---

- ▶  $\tau = \{E\}$ , with  $E$  binary.

# Examples of Structures : A Graph

---

- ▶  $\tau = \{E\}$ , with  $E$  binary.
  - ▶ A graph structure over  $\tau$  is  $\mathcal{G} = (V, E^{\mathcal{G}})$ ,
  - ▶ The **universe**  $u(\mathcal{G})$  is the set of vertices  $V$
  - ▶ The relation  $E$  is the edge relation

# Examples of Structures : A Graph

---

- ▶  $\tau = \{E\}$ , with  $E$  binary.
  - ▶ A graph structure over  $\tau$  is  $\mathcal{G} = (V, E^{\mathcal{G}})$ ,
  - ▶ The **universe**  $u(\mathcal{G})$  is the set of vertices  $V$
  - ▶ The relation  $E$  is the edge relation
  - ▶  $\mathcal{G} = (V = \{1, 2, 3, 4\}, E^{\mathcal{G}} = \{(1, 2), (2, 3), (3, 4), (1, 1)\})$ . We could just as well draw the graph for convenience.

# Examples of Structures : An Order

---

- ▶  $\tau = \{<, S\}$  with  $<$ ,  $S$  binary.

# Examples of Structures : An Order

---

- ▶  $\tau = \{<, S\}$  with  $<$ ,  $S$  binary.
  - ▶ A finite order structure over  $\tau$  is  $\mathcal{O} = (O, <^{\mathcal{O}}, S^{\mathcal{O}})$
  - ▶ The universe  $u(\mathcal{O})$  is the finite ordered set  $O$
  - ▶  $<^{\mathcal{O}}$  is the ordering on  $O$  and  $S^{\mathcal{O}}$  is the successor on  $O$

# Examples of Structures : An Order

---

- ▶  $\tau = \{<, S\}$  with  $<$ ,  $S$  binary.
  - ▶ A finite order structure over  $\tau$  is  $\mathcal{O} = (O, <^{\mathcal{O}}, S^{\mathcal{O}})$
  - ▶ The universe  $u(\mathcal{O})$  is the finite ordered set  $O$
  - ▶  $<^{\mathcal{O}}$  is the ordering on  $O$  and  $S^{\mathcal{O}}$  is the successor on  $O$
  - ▶  $O = (O = \{1, 2, 4\}, <^{\mathcal{O}} = \{(1, 2), (1, 4), (2, 4)\}, S^{\mathcal{O}} = \{(1, 2)\})$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a$ ,  $Q_b$  are unary relations.

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a$ ,  $Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a, Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a, Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$
  - ▶  $<^{\mathcal{W}}$  is the ordering relation on the positions of  $W$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a, Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$
  - ▶  $<^{\mathcal{W}}$  is the ordering relation on the positions of  $W$
  - ▶  $S^{\mathcal{W}}$  is the successor relation on the positions of  $W$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a$ ,  $Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$
  - ▶  $<^{\mathcal{W}}$  is the ordering relation on the positions of  $W$
  - ▶  $S^{\mathcal{W}}$  is the successor relation on the positions of  $W$
  - ▶  $Q_a{}^{\mathcal{W}}$  is the set of positions labeled  $a$  in  $W$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a$ ,  $Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$
  - ▶  $<^{\mathcal{W}}$  is the ordering relation on the positions of  $W$
  - ▶  $S^{\mathcal{W}}$  is the successor relation on the positions of  $W$
  - ▶  $Q_a{}^{\mathcal{W}}$  is the set of positions labeled  $a$  in  $W$
  - ▶  $Q_b{}^{\mathcal{W}}$  is the set of positions labeled  $b$  in  $W$

# Examples of Structures : A Word

---

- ▶  $\tau = \{<, S, Q_a, Q_b\}$ , where  $<$ ,  $S$  are binary,  $Q_a$ ,  $Q_b$  are unary relations.
  - ▶ A word structure  $\mathcal{W} = (u(\mathcal{W}), <^{\mathcal{W}}, S^{\mathcal{W}}, Q_a{}^{\mathcal{W}}, Q_b{}^{\mathcal{W}})$
  - ▶ The universe  $u(\mathcal{W})$  consists of the positions in a word  $W$  over symbols  $a, b$
  - ▶  $<^{\mathcal{W}}$  is the ordering relation on the positions of  $W$
  - ▶  $S^{\mathcal{W}}$  is the successor relation on the positions of  $W$
  - ▶  $Q_a{}^{\mathcal{W}}$  is the set of positions labeled  $a$  in  $W$
  - ▶  $Q_b{}^{\mathcal{W}}$  is the set of positions labeled  $b$  in  $W$
  - ▶ The structure with  $u(\mathcal{W}) = \{0, 1, 2, \dots, 8\}$ ,  
 $Q_a{}^{\mathcal{W}} = \{0, 1, 4, 6, 8\}$ ,  $Q_b{}^{\mathcal{W}} = \{2, 3, 5, 7\}$ ,
  - ▶  $<^{\mathcal{W}} = \{(0, 1), (0, 2), \dots, (7, 8)\}$ ,  $S^{\mathcal{W}} = \{(0, 1), (1, 2), \dots, (7, 8)\}$   
uniquely defines the word  $W = aabbababa$ .
  - ▶ For convenience, we can just write the word instead of the structure.

# Free and Bound Variables

---

- ▶ For a wff  $\varphi = \forall x\psi$  or  $\exists x\psi$ ,  $\psi$  is said to be the **scope** of the quantifier  $x$

# Free and Bound Variables

---

- ▶ For a wff  $\varphi = \forall x\psi$  or  $\exists x\psi$ ,  $\psi$  is said to be the **scope** of the quantifier  $x$
- ▶ Every occurrence of  $x$  in  $\forall x\psi$  or  $\exists x\psi$  is **bound**
- ▶ Any occurrence of  $x$  which is not bound is called **free**

# Free and Bound Variables

---

- ▶ For a wff  $\varphi = \forall x\psi$  or  $\exists x\psi$ ,  $\psi$  is said to be the **scope** of the quantifier  $x$
- ▶ Every occurrence of  $x$  in  $\forall x\psi$  or  $\exists x\psi$  is **bound**
- ▶ Any occurrence of  $x$  which is not bound is called **free**
- ▶  $\varphi = P(x, y) \rightarrow \forall x((\forall y R(x, y)) \rightarrow Q(x, y))$ 
  - ▶  $y$  is free in  $Q(x, y)$  and bound in  $R(x, y)$ ,
  - ▶  $x$  is free in  $P(x, y)$ , and bound in  $Q(x, y), R(x, y)$

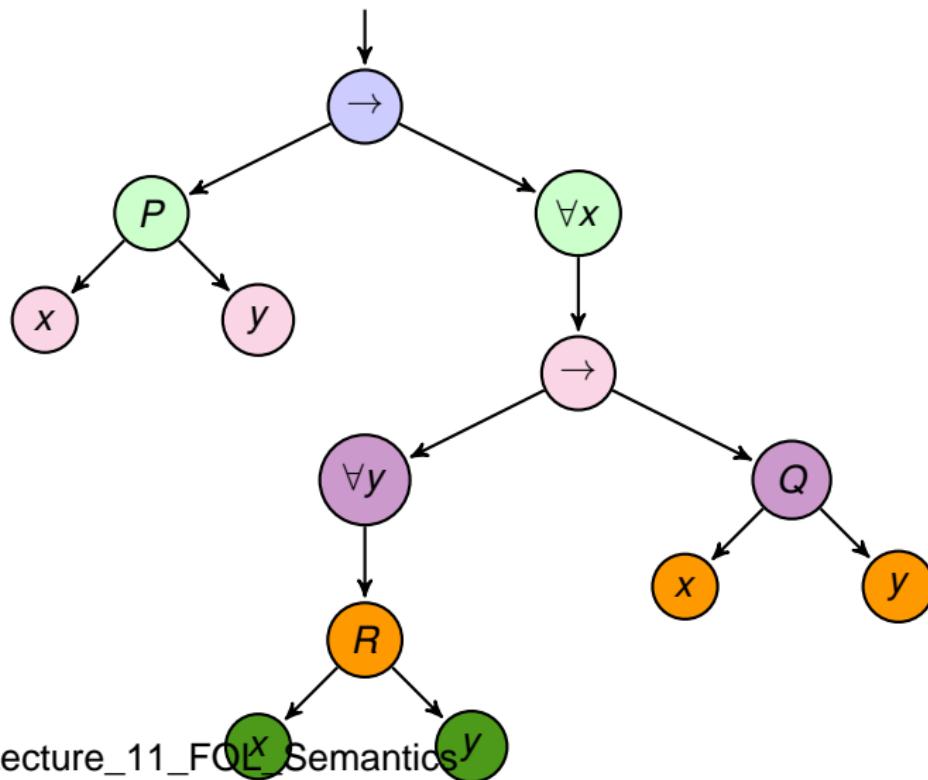
# Free and Bound Variables

---

- ▶ For a wff  $\varphi = \forall x\psi$  or  $\exists x\psi$ ,  $\psi$  is said to be the **scope** of the quantifier  $x$
- ▶ Every occurrence of  $x$  in  $\forall x\psi$  or  $\exists x\psi$  is **bound**
- ▶ Any occurrence of  $x$  which is not bound is called **free**
- ▶  $\varphi = P(x, y) \rightarrow \forall x((\forall y R(x, y)) \rightarrow Q(x, y))$ 
  - ▶  $y$  is free in  $Q(x, y)$  and bound in  $R(x, y)$ ,
  - ▶  $x$  is free in  $P(x, y)$ , and bound in  $Q(x, y), R(x, y)$
- ▶ Given  $\varphi$ , denote by  $\varphi(x_1, \dots, x_n)$ , that  $x_1, \dots, x_n$  are the free variables of  $\varphi$ , also *free*( $\varphi$ )
- ▶ A **sentence** is a formula  $\varphi$  **none** of whose variables are **free**

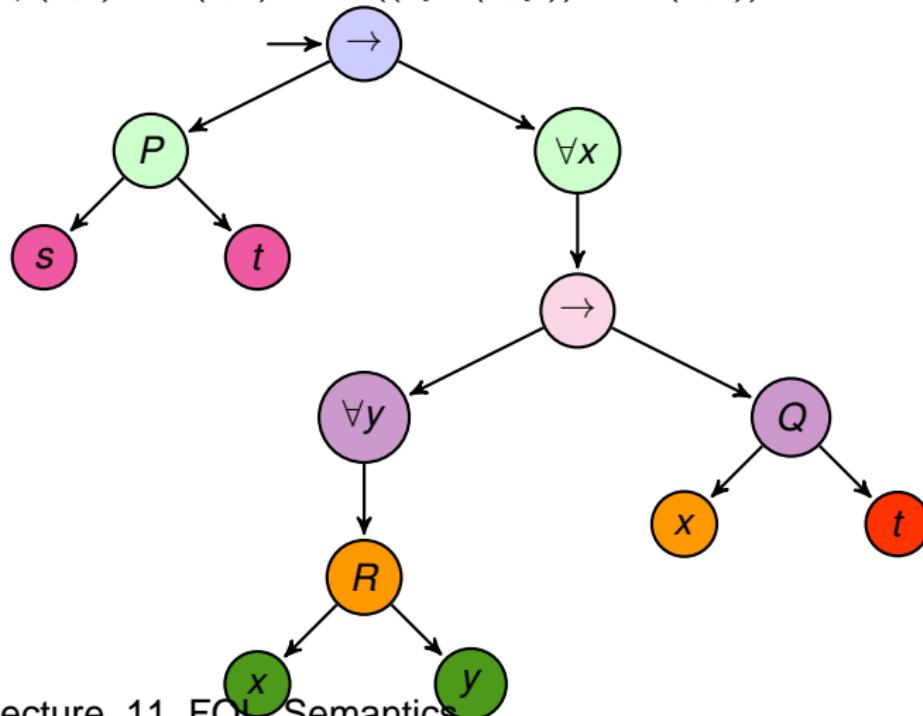
$$P(x, y) \rightarrow \forall x((\forall y R(x, y)) \rightarrow Q(x, y))$$

---



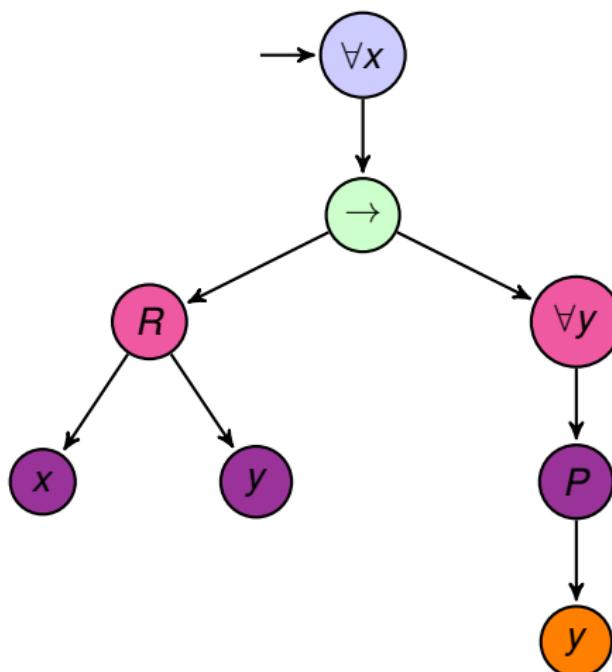
$$P(x, y) \rightarrow \forall x((\forall y R(x, y)) \rightarrow Q(x, y))$$

$$\varphi(s, t) = P(s, t) \rightarrow \forall x((\forall y R(x, y)) \rightarrow Q(x, t))$$



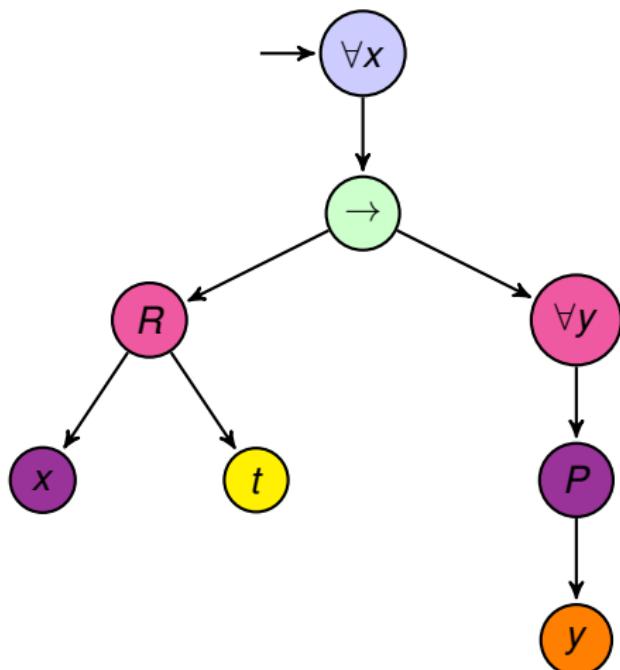
$$\forall x(R(x, y) \rightarrow \forall y P(y))$$

---



$$\forall x(R(x, y) \rightarrow \forall y P(y))$$

---



$$\varphi(t) = \forall x(R(x, t) \rightarrow \forall y P(y))$$

Lecture\_11\_FOL\_Semantics

# Assignments on $\tau$ -structures

---

## Assignments

For a  $\tau$ -structure  $\mathcal{A}$ , an assignment over  $\mathcal{A}$  is a function  $\alpha : \mathcal{V} \rightarrow u(\mathcal{A})$  that assigns every variable  $x \in \mathcal{V}$  a value  $\alpha(x) \in u(\mathcal{A})$ . If  $t$  is a constant symbol  $c$ , then  $\alpha(t)$  is  $c^{\mathcal{A}}$

# Assignments on $\tau$ -structures

## Assignments

For a  $\tau$ -structure  $\mathcal{A}$ , an assignment over  $\mathcal{A}$  is a function  $\alpha : \mathcal{V} \rightarrow u(\mathcal{A})$  that assigns every variable  $x \in \mathcal{V}$  a value  $\alpha(x) \in u(\mathcal{A})$ . If  $t$  is a constant symbol  $c$ , then  $\alpha(t)$  is  $c^{\mathcal{A}}$

## Binding on a Variable

For an assignment  $\alpha$  over  $\mathcal{A}$ ,  $\alpha[x \mapsto a]$  is the assignment

$$\alpha[x \mapsto a](y) = \begin{cases} \alpha(y), & y \neq x, \\ a, & y = x \end{cases}$$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- $\mathcal{A} \not\models_{\alpha} \perp$
- $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$
- ▶  $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$
- ▶  $\mathcal{A} \models_{\alpha} R(t_1, \dots, t_k)$  iff  $(\alpha(t_1), \dots, \alpha(t_k)) \in R^{\mathcal{A}}$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$
- ▶  $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$
- ▶  $\mathcal{A} \models_{\alpha} R(t_1, \dots, t_k)$  iff  $(\alpha(t_1), \dots, \alpha(t_k)) \in R^{\mathcal{A}}$
- ▶  $\mathcal{A} \models_{\alpha} (\varphi \rightarrow \psi)$  iff  $\mathcal{A} \not\models_{\alpha} \varphi$  or  $\mathcal{A} \models_{\alpha} \psi$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$
- ▶  $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$
- ▶  $\mathcal{A} \models_{\alpha} R(t_1, \dots, t_k)$  iff  $(\alpha(t_1), \dots, \alpha(t_k)) \in R^{\mathcal{A}}$
- ▶  $\mathcal{A} \models_{\alpha} (\varphi \rightarrow \psi)$  iff  $\mathcal{A} \not\models_{\alpha} \varphi$  or  $\mathcal{A} \models_{\alpha} \psi$
- ▶  $\mathcal{A} \models_{\alpha} (\forall x)\varphi$  iff for every  $a \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} \varphi$

# Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$
- ▶  $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$
- ▶  $\mathcal{A} \models_{\alpha} R(t_1, \dots, t_k)$  iff  $(\alpha(t_1), \dots, \alpha(t_k)) \in R^{\mathcal{A}}$
- ▶  $\mathcal{A} \models_{\alpha} (\varphi \rightarrow \psi)$  iff  $\mathcal{A} \not\models_{\alpha} \varphi$  or  $\mathcal{A} \models_{\alpha} \psi$
- ▶  $\mathcal{A} \models_{\alpha} (\forall x)\varphi$  iff for every  $a \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} \varphi$
- ▶  $\mathcal{A} \models_{\alpha} (\exists x)\varphi$  iff there is some  $a \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} \varphi$

Last two cases,  $\alpha$  has no effect on the value of  $x$ . Thus, assignments matter **only** to free variables.

# CS 228 : Logic in Computer Science

Krishna. S

# Recap

---

Signatures, Formulae over signatures, Structure for a signature

# Example of Satisfaction

---

- ▶  $\mathcal{G} = (\{1, 2, 3\}, E^{\mathcal{G}} = \{(1, 2), (2, 1), (2, 3), (3, 2)\})$ 
  - ▶ For any assignment  $\alpha$ ,  $\mathcal{G} \models_{\alpha} \forall x \forall y (E(x, y) \rightarrow E(y, x))$  iff for every  $a, b \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a, y \mapsto b]} (E(x, y) \rightarrow E(y, x))$

# Example of Satisfaction

---

- ▶  $\mathcal{G} = (\{1, 2, 3\}, E^{\mathcal{G}} = \{(1, 2), (2, 1), (2, 3), (3, 2)\})$ 
  - ▶ For any assignment  $\alpha$ ,  $\mathcal{G} \models_{\alpha} \forall x \forall y (E(x, y) \rightarrow E(y, x))$  iff for every  $a, b \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a, y \mapsto b]} (E(x, y) \rightarrow E(y, x))$
  - ▶ There is an assignment  $\alpha$  which satisfies  
 $\mathcal{G} \models_{\alpha} \exists x (E(x, y) \wedge E(x, z) \wedge y \neq z)$

# Example of Satisfaction

---

- ▶  $\mathcal{G} = (\{1, 2, 3\}, E^{\mathcal{G}} = \{(1, 2), (2, 1), (2, 3), (3, 2)\})$ 
  - ▶ For any assignment  $\alpha$ ,  $\mathcal{G} \models_{\alpha} \forall x \forall y (E(x, y) \rightarrow E(y, x))$  iff for every  $a, b \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a, y \mapsto b]} (E(x, y) \rightarrow E(y, x))$
  - ▶ There is an assignment  $\alpha$  which satisfies  $\mathcal{G} \models_{\alpha} \exists x (E(x, y) \wedge E(x, z) \wedge y \neq z)$
  - ▶ There is no assignment  $\alpha$  which satisfies  $\exists x \forall y (E(x, y))$
- ▶  $\mathcal{W} = abaaa$

# Example of Satisfaction

---

- ▶  $\mathcal{G} = (\{1, 2, 3\}, E^{\mathcal{G}} = \{(1, 2), (2, 1), (2, 3), (3, 2)\})$ 
  - ▶ For any assignment  $\alpha$ ,  $\mathcal{G} \models_{\alpha} \forall x \forall y (E(x, y) \rightarrow E(y, x))$  iff for every  $a, b \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a, y \mapsto b]} (E(x, y) \rightarrow E(y, x))$
  - ▶ There is an assignment  $\alpha$  which satisfies  
 $\mathcal{G} \models_{\alpha} \exists x (E(x, y) \wedge E(x, z) \wedge y \neq z)$
  - ▶ There is no assignment  $\alpha$  which satisfies  $\exists x \forall y (E(x, y))$
- ▶  $\mathcal{W} = abaaa$ 
  - ▶ There is an assignment  $\alpha$  for which  
 $\mathcal{W} \models_{\alpha} (Q_a(x) \wedge Q_a(y) \wedge S(x, y))$

# Example of Satisfaction

---

- ▶  $\mathcal{G} = (\{1, 2, 3\}, E^{\mathcal{G}} = \{(1, 2), (2, 1), (2, 3), (3, 2)\})$ 
  - ▶ For any assignment  $\alpha$ ,  $\mathcal{G} \models_{\alpha} \forall x \forall y (E(x, y) \rightarrow E(y, x))$  iff for every  $a, b \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a, y \mapsto b]} (E(x, y) \rightarrow E(y, x))$
  - ▶ There is an assignment  $\alpha$  which satisfies  
 $\mathcal{G} \models_{\alpha} \exists x (E(x, y) \wedge E(x, z) \wedge y \neq z)$
  - ▶ There is no assignment  $\alpha$  which satisfies  $\exists x \forall y (E(x, y))$
- ▶  $\mathcal{W} = abaaa$ 
  - ▶ There is an assignment  $\alpha$  for which  
 $\mathcal{W} \models_{\alpha} (Q_a(x) \wedge Q_a(y) \wedge S(x, y))$
  - ▶ There is no assignment  $\alpha$  which satisfies  
 $\exists x \exists y (Q_b(x) \wedge Q_b(y) \wedge x \neq y)$

# Satisfiability, Validity and Equivalence

---

- ▶ A formula  $\varphi$  over a signature  $\tau$  is said to be **satisfiable** iff for some  $\tau$ -structure  $\mathcal{A}$  and assignment  $\alpha$ ,  $\mathcal{A} \models_\alpha \varphi$

# Satisfiability, Validity and Equivalence

---

- ▶ A formula  $\varphi$  over a signature  $\tau$  is said to be **satisfiable** iff for some  $\tau$ -structure  $\mathcal{A}$  and assignment  $\alpha$ ,  $\mathcal{A} \models_{\alpha} \varphi$
- ▶ A formula  $\varphi$  over a signature  $\tau$  is said to be **valid** iff for every  $\tau$ -structure  $\mathcal{A}$  and assignment  $\alpha$ ,  $\mathcal{A} \models_{\alpha} \varphi$

# Satisfiability, Validity and Equivalence

---

- ▶ A formula  $\varphi$  over a signature  $\tau$  is said to be **satisfiable** iff for some  $\tau$ -structure  $\mathcal{A}$  and assignment  $\alpha$ ,  $\mathcal{A} \models_{\alpha} \varphi$
- ▶ A formula  $\varphi$  over a signature  $\tau$  is said to be **valid** iff for every  $\tau$ -structure  $\mathcal{A}$  and assignment  $\alpha$ ,  $\mathcal{A} \models_{\alpha} \varphi$
- ▶ Formulae  $\varphi(x_1, \dots, x_n)$  and  $\psi(x_1, \dots, x_n)$  are **equivalent** denoted  $\varphi \equiv \psi$  iff for every  $\mathcal{A}$  and  $\alpha$ ,  $\mathcal{A} \models_{\alpha} \varphi$  iff  $\mathcal{A} \models_{\alpha} \psi$

# Equisatisfiability

---

Let  $\varphi_1(x) = \forall y R(x, y)$  and  $\varphi_2 = \exists x \forall y R(x, y)$ .

- ▶ It is clear that whenever  $\mathcal{A} \models \varphi_2$ , one can find an assignment  $\alpha$  such that  $\mathcal{A} \models_{\alpha} \varphi_1(x)$ .
- ▶ Likewise, if  $\mathcal{A} \models_{\alpha} \varphi_1(x)$ , then  $\mathcal{A} \models \varphi_2$ .
- ▶ Thus,  $\varphi_1(x), \varphi_2$  are **equisatisfiable**.

# True or False?

---

For a formula  $\varphi$  and assignments  $\alpha_1$  and  $\alpha_2$  such that for every  $x \in \text{free}(\varphi)$ ,  $\alpha_1(x) = \alpha_2(x)$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

# True or False?

---

For a formula  $\varphi$  and assignments  $\alpha_1$  and  $\alpha_2$  such that for every  $x \in \text{free}(\varphi)$ ,  $\alpha_1(x) = \alpha_2(x)$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

- ▶ For example,  $\varphi(y) = \forall x(R(x, y) \rightarrow \forall zP(z))$

# True or False?

---

For a formula  $\varphi$  and assignments  $\alpha_1$  and  $\alpha_2$  such that for every  $x \in \text{free}(\varphi)$ ,  $\alpha_1(x) = \alpha_2(x)$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

- ▶ For example,  $\varphi(y) = \forall x(R(x, y) \rightarrow \forall zP(z))$
- ▶ Consider two assignments  $\alpha_1, \alpha_2$  such that  $\alpha_1(y) = \alpha_2(y) = \alpha$  (say)

# True or False?

---

For a formula  $\varphi$  and assignments  $\alpha_1$  and  $\alpha_2$  such that for every  $x \in \text{free}(\varphi)$ ,  $\alpha_1(x) = \alpha_2(x)$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

- ▶ For example,  $\varphi(y) = \forall x(R(x, y) \rightarrow \forall z P(z))$
- ▶ Consider two assignments  $\alpha_1, \alpha_2$  such that  $\alpha_1(y) = \alpha_2(y) = \alpha$  (say)
- ▶ Evaluate for all  $a, b \in u(\mathcal{A})$ ,  $R(a, \alpha) \rightarrow P(b)$

# True or False?

---

For a formula  $\varphi$  and assignments  $\alpha_1$  and  $\alpha_2$  such that for every  $x \in \text{free}(\varphi)$ ,  $\alpha_1(x) = \alpha_2(x)$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

- ▶ For example,  $\varphi(y) = \forall x(R(x, y) \rightarrow \forall zP(z))$
- ▶ Consider two assignments  $\alpha_1, \alpha_2$  such that  $\alpha_1(y) = \alpha_2(y) = \alpha$  (say)
- ▶ Evaluate for all  $a, b \in u(\mathcal{A})$ ,  $R(a, \alpha) \rightarrow P(b)$
- ▶  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  $\mathcal{A} \models_{\alpha_2} \varphi$

# True or False?

---

For a sentence  $\varphi$ , and any two assignments  $\alpha_1$  and  $\alpha_2$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  
 $\mathcal{A} \models_{\alpha_2} \varphi$

# True or False?

---

For a sentence  $\varphi$ , and any two assignments  $\alpha_1$  and  $\alpha_2$ ,  $\mathcal{A} \models_{\alpha_1} \varphi$  iff  
 $\mathcal{A} \models_{\alpha_2} \varphi$

No free variables!

# Check Satisfiability

---

Let  $\tau$  be a signature with a single unary relation  $P$ . Consider the structure  $\mathcal{A} = (U_{\mathcal{A}} = \{0, 1\}, P^{\mathcal{A}} = \{1\})$ .

Let  $\varphi = \forall x_1 \forall x_2 \dots \forall x_n (P(x_1) \rightarrow (P(x_2) \rightarrow (P(x_3) \dots \rightarrow (P(x_n) \rightarrow P(x_1)) \dots)))$ .

Does  $\mathcal{A} \models \varphi$ ?

# Check Satisfiability

---

Let  $\varphi(y) = \exists x(E(x, y) \wedge \neg(y = x) \wedge \forall z[E(z, y) \rightarrow z = x])$  over the signature  $\tau$  containing a binary relation  $E$ . Is  $\varphi(y)$  satisfiable under some graph structure?

# CS 228 : Logic in Computer Science

Krishna. S

# Check Satisfiability

---

Let  $\psi(z) = \exists x [Q_a(x) \wedge \forall y [(y \leq x \wedge Q_b(y)) \rightarrow (z < x \wedge y < z \wedge Q_c(z))]]$   
over the signature  $\tau$  having the relational symbols  $<$ ,  $Q_a$ ,  $Q_b$ ,  $Q_c$  and  
unary function  $S$ . Does  $\psi(z)$  evaluate to true under some word  
structure?

# Check Satisfiability

---

Let  $\zeta = P(0) \wedge \forall x(P(x) \rightarrow P(S(x))) \wedge \exists x \neg P(x)$  over a signature  $\tau$  containing the constant 0, unary function  $S$  and unary relation  $P$ .  
Is  $\zeta$  satisfiable?

## Normal Forms in FOL

# Recap : Satisfaction

---

We define the relation  $\mathcal{A} \models_{\alpha} \varphi$  (read as  $\varphi$  is true in  $\mathcal{A}$  under the assignment  $\alpha$ ) inductively:

- ▶  $\mathcal{A} \not\models_{\alpha} \perp$
- ▶  $\mathcal{A} \models_{\alpha} t_1 = t_2$  iff  $\alpha(t_1) = \alpha(t_2)$
- ▶  $\mathcal{A} \models_{\alpha} R(t_1, \dots, t_k)$  iff  $(\alpha(t_1), \dots, \alpha(t_k)) \in R^{\mathcal{A}}$
- ▶  $\mathcal{A} \models_{\alpha} (\varphi \rightarrow \psi)$  iff  $\mathcal{A} \not\models_{\alpha} \varphi$  or  $\mathcal{A} \models_{\alpha} \psi$
- ▶  $\mathcal{A} \models_{\alpha} (\forall x)\varphi$  iff for every  $a \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} \varphi$
- ▶  $\mathcal{A} \models_{\alpha} (\exists x)\varphi$  iff there is some  $a \in u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} \varphi$

Last two cases,  $\alpha$  has no effect on the value of  $x$ . Thus, assignments matter **only** to free variables.

# Equivalences

---

Let  $F, G$  be arbitrary FOL formulae.

1.  $\neg \forall x F \equiv \exists x \neg F$
2.  $\neg \exists x F \equiv \forall x \neg F$

$\mathcal{A} \models_{\alpha} \neg \forall x F$  iff  $\mathcal{A} \not\models_{\alpha} \forall x F$   
iff  $\mathcal{A} \not\models_{\alpha[x \mapsto a]} F$  for some  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[x \mapsto a]} \neg F$  for some  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha} \exists x \neg F$

# Equivalences

---

If  $x$  does not occur free in  $G$  then

1.  $(\forall x F \wedge G) \equiv \forall x(F \wedge G)$
2.  $(\forall x F \vee G) \equiv \forall x(F \vee G)$
3.  $(\exists x F \wedge G) \equiv \exists x(F \wedge G)$
4.  $(\exists x F \vee G) \equiv \exists x(F \vee G)$

$\mathcal{A} \models_{\alpha} \forall x F \wedge G$  iff  $\mathcal{A} \models_{\alpha} \forall x F$  and  $\mathcal{A} \models_{\alpha} G$

iff for all  $a \in U^{\mathcal{A}}$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} F$  and  $\mathcal{A} \models_{\alpha} G$

iff for all  $a \in U^{\mathcal{A}}$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} F$  and  $\mathcal{A} \models_{\alpha[x \mapsto a]} G$

iff for all  $a \in U^{\mathcal{A}}$ ,  $\mathcal{A} \models_{\alpha[x \mapsto a]} (F \wedge G)$

iff  $\mathcal{A} \models \forall x(F \wedge G)$

# Equivalences

---

Let  $F, G$  be arbitrary FOL formulae.

$$1. (\forall x F \wedge \forall x G) \equiv \forall x (F \wedge G)$$

$$2. (\exists x F \vee \exists x G) \equiv \exists x (F \vee G)$$

$$1. \forall x \forall y F \equiv \forall y \forall x F$$

$$2. \exists x \exists y F \equiv \exists y \exists x F$$

# Recap : Terms

---

Given a signature  $\tau$ , the set of  $\tau$ -terms are defined inductively as follows.

- ▶ Each variable is a term
- ▶ Each constant symbol is a term
- ▶ If  $t_1, \dots, t_k$  are terms and  $f$  is a  $k$ -ary function, then  $f(t_1, \dots, t_k)$  is a term
- ▶ Ground Terms : Terms without variables. For instance  $f(c_1, \dots, c_k)$  for constants  $c_1, \dots, c_k$ .

# Translation Lemma

---

## Translation Lemma

If  $t$  is a term and  $F$  is a formula such that no variable in  $t$  occurs bound in  $F$ , then  $\mathcal{A} \models_{\alpha} F[t/x]$  iff  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} F$ .

$F[t/x]$  denotes substituting  $t$  for  $x$  in  $F$ , where  $x$  is free in  $F$

- ▶ What if  $t$  contains a variable bound in  $F$ ?
- ▶ Results in *Variable Capture*

# Translation Lemma Proof : Optional

---

Proof by Induction on formulae.

- ▶ Base case. Atomic formulae  $P(t_1, \dots, t_k)$ .
- ▶  $\mathcal{A} \models_{\alpha} P(t_1, \dots, t_k)[t/x]$  iff  $\mathcal{A} \models_{\alpha} P(t_1[t/x], \dots, t_k[t/x])$ .
- ▶ Show that  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} P(t_1, \dots, t_k)$ .
  - ▶ Base Cases within :  $t_i = c$ ,  $t_i = y$  for  $y \neq x$ ,  $t_i = x$  for each  $t_i$ .
  - ▶ Case  $t_i = f(s_1, \dots, s_j)$  for a function  $f$ .
  - ▶  $f(s_1, \dots, s_j)[t/x] = f(s_1[t/x], \dots, s_j[t/x])$
- ▶  $\mathcal{A} \models_{\alpha} P(t_1[t/x], \dots, t_k[t/x])$  iff  $(\alpha(t_1[t/x]), \dots, \alpha(t_k[t/x])) \in P^{\mathcal{A}}$
- ▶ iff  $(\alpha([x \mapsto \alpha(t)](t_1), \dots, \alpha([x \mapsto \alpha(t)](t_k)) \in P^{\mathcal{A}}$
- ▶ iff  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} P(t_1, \dots, t_k)$
- ▶ Cases for formulae with propositional connectives is routine.
- ▶ Case with quantifier,  $\forall y F[t/x]$ ,  $\exists y F[t/x]$  where  $y \neq x$ .

# Renaming

---

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$

# Renaming

---

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_\alpha \forall x G$

# CS 228 : Logic in Computer Science

Krishna. S

## Normal Forms in FOL

# Translation Lemma

---

## Translation Lemma

If  $t$  is a term and  $F$  is a formula such that no variable in  $t$  occurs bound in  $F$ , then  $\mathcal{A} \models_{\alpha} F[t/x]$  iff  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} F$ .

$F[t/x]$  denotes substituting  $t$  for  $x$  in  $F$ , where  $x$  is free in  $F$

- ▶ What if  $t$  contains a variable bound in  $F$ ?
- ▶ Results in *Variable Capture*

# Translation Lemma Proof : Optional

---

Proof by Induction on formulae.

- ▶ Base case. Atomic formulae  $P(t_1, \dots, t_k)$ .
- ▶  $\mathcal{A} \models_{\alpha} P(t_1, \dots, t_k)[t/x]$  iff  $\mathcal{A} \models_{\alpha} P(t_1[t/x], \dots, t_k[t/x])$ .
- ▶ Show that  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} P(t_1, \dots, t_k)$ .
  - ▶ Base Cases within :  $t_i = c$ ,  $t_i = y$  for  $y \neq x$ ,  $t_i = x$  for each  $t_i$ .
  - ▶ Case  $t_i = f(s_1, \dots, s_j)$  for a function  $f$ .
  - ▶  $f(s_1, \dots, s_j)[t/x] = f(s_1[t/x], \dots, s_j[t/x])$
- ▶  $\mathcal{A} \models_{\alpha} P(t_1[t/x], \dots, t_k[t/x])$  iff  $(\alpha(t_1[t/x]), \dots, \alpha(t_k[t/x])) \in P^{\mathcal{A}}$
- ▶ iff  $(\alpha([x \mapsto \alpha(t)](t_1), \dots, \alpha([x \mapsto \alpha(t)](t_k)) \in P^{\mathcal{A}}$
- ▶ iff  $\mathcal{A} \models_{\alpha[x \mapsto \alpha(t)]} P(t_1, \dots, t_k)$
- ▶ Cases for formulae with propositional connectives is routine.
- ▶ Case with quantifier,  $\forall y F[t/x]$ ,  $\exists y F[t/x]$  where  $y \neq x$ .

# Renaming

---

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$

# Renaming

---

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$

# Renaming

$\int_0^\infty f(s)ds$  has the same value as  $\int_0^\infty f(t)dt$

## Renaming Lemma

Let  $F = Qx[G]$  be a formula with  $Q \in \{\exists, \forall\}$ . Let  $y$  be a variable which does not appear in  $G$ . Then  $\mathcal{A} \models_\alpha F$  iff  $\mathcal{A} \models_\alpha Qy(G[y/x])$ .

Assume  $Q = \forall$ .

$\mathcal{A} \models_\alpha \forall y G[y/x]$  iff  $\mathcal{A} \models_{\alpha[y \mapsto a]} G[y/x]$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto \alpha[y \mapsto a](y)]} G$  for all  $a \in U^{\mathcal{A}}$   
(Translation Lemma)  
iff  $\mathcal{A} \models_{\alpha[y \mapsto a, x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_{\alpha[x \mapsto a]} G$  for all  $a \in U^{\mathcal{A}}$   
iff  $\mathcal{A} \models_\alpha \forall x G$

# Rectified Formulae

---

A FOL formula is *rectified* if no variable occurs both free and bound, and if all quantifiers in the formula refer to different variables.

$$\forall x \exists y P(x, f(y)) \wedge \forall y (Q(x, y) \vee R(x))$$

is not rectified. By renaming we obtain an equivalent rectified formula

$$\forall u \exists v P(u, f(v)) \wedge \forall y (Q(x, y) \vee R(x))$$

By Renaming Lemma, we can always obtain an equivalent rectified formula by renaming bound variables.

# Prenex Normal Form

---

A formula is in prenex normal form if it can be written as

$$Q_1 x_1 Q_2 x_2 \dots Q_n x_n F$$

where  $Q_i \in \{\forall, \exists\}$ ,  $n \geq 0$  and  $F$  has no quantifiers.  $F$  is called the matrix of the formula.

# Prenex Normal Form : Example

---

Convert the rectified formula  $\neg(\exists xP(x, y) \vee \forall zQ(z)) \wedge \exists wQ(w)$  to Prenex Normal Form

- ▶  $(\neg\exists xP(x, y) \wedge \neg\forall zQ(z)) \wedge \exists wQ(w)$
- ▶  $(\forall x\neg P(x, y) \wedge \exists z\neg Q(z)) \wedge \exists wQ(w)$
- ▶  $\forall x\exists z(\neg P(x, y) \wedge \neg Q(z)) \wedge \exists wQ(w)$
- ▶  $\forall x\exists z\exists w((\neg P(x, y) \wedge \neg Q(z)) \wedge Q(w))$
- ▶ Note that we have used the equivalences from the last lecture

# Rectified, Prenex normal form (RPF)

---

- ▶ Given a rectified formula, we can use the equivalences from the last lecture to convert  $F$  into rectified, prenex normal form, by “pushing all quantifiers up front”.
- ▶ Otherwise, rectify the formula first, and then convert to prenex normal form.

Every formula is equivalent to a rectified formula in prenex normal form.

# Skolemisation

---

A formula in RPF is in *Skolem form* if it has no occurrences of the existential quantifier.

We can transform any formula in RPF to an equisatisfiable formula in Skolem form by using extra function symbols.

- ▶  $\forall x \exists y P(x, y)$  is equisatisfiable with  $\forall x P(x, f(x))$ .
- ▶  $\forall x \forall z \exists y P(x, y, z)$  is equisatisfiable with  $\forall x \forall z P(x, f(x, z), z)$ .
- ▶  $\exists x \forall y G(x, y)$  is equisatisfiable with  $\forall y G(c, y)$  where  $c$  is a constant.
- ▶  $\exists x \forall y \exists z \exists w G(x, y, z, w)$  is equisatisfiable with  $\forall y G(c, y, f(y), g(y))$  where  $c$  is a constant.

# Skolemisation

## Skolem Lemma

Let  $F = \forall y_1 \forall y_2 \dots \forall y_n \exists z G$  be in RPF. Given a function symbol  $f$  of arity  $n$  which does not appear in  $F$ , write

$$F' = \forall y_1 \forall y_2 \dots \forall y_n G[f(y_1, \dots, y_n)/z]$$

Then  $F$  and  $F'$  are equisatisfiable.

Assume  $F$  is satisfiable. Let  $\mathcal{A} \models_{\alpha} F$ .

- ▶ Extend structure  $\mathcal{A}$  with an interpretation for a function  $f$  such that  $\mathcal{A}' \models_{\alpha'} F'$ .
- ▶ Given  $a_1, \dots, a_n \in U^{\mathcal{A}}$ , choose  $a \in U^{\mathcal{A}}$  such that  $\mathcal{A} \models_{\alpha[y_1 \mapsto a_1, \dots, y_n \mapsto a_n, z \mapsto a]} G$ , and define  $f^{\mathcal{A}'}(a_1, \dots, a_n) = a$ .
- ▶  $f$  does not appear in  $G$ ,  $\mathcal{A}' \models_{\alpha[y_1 \mapsto a_1, \dots, y_n \mapsto a_n, z \mapsto f^{\mathcal{A}'}(a_1, \dots, a_n)]} G$ ,
- ▶ By Translation Lemma,  $\mathcal{A}' \models_{\alpha[y_1 \mapsto a_1, \dots, y_n \mapsto a_n]} G[f(y_1, \dots, y_n)/z]$
- ▶ Since this holds for any  $a_1, \dots, a_n \in U^{\mathcal{A}}$ ,  
 $\mathcal{A}' \models \forall y_1 \forall y_2 \dots \forall y_n G[f(y_1, \dots, y_n)/z]$

# Skolemisation : Example

---

$$\forall x \exists y \forall z \exists w (\neg P(a, w) \vee Q(f(x), y))$$

- ▶ By Skolem Lemma, eliminate  $\exists y$  and introduce a new function  $g$ , obtaining  $\forall x \forall z \exists w (\neg P(a, w) \vee Q(f(x), g(x)))$
- ▶ By Skolem Lemma, eliminate  $\exists w$  introducing a new function  $h$  obtaining  $\forall x \forall z (\neg P(a, h(x, z)) \vee Q(f(x), g(x)))$

# Conversion to Skolem Form : Summary

---

Convert an arbitrary FOL formula to an equisatisfiable formula in Skolem formula as follows:

1. Rectify  $F$  systematically renaming bound variables, obtaining an equivalent formula  $F_1$
2. Use the equivalences in the beginning and move all quantifiers outside, yielding an equivalent formula  $F_2$  in prenex normal form
3. Repeatedly eliminate the outermost existential quantifier in  $F_2$  until an equisatisfiable formula  $F_3$  is obtained in Skolem form.

# Semi Decidability of Satisfiability

---

- ▶ Given a FOL formula in Skolem normal form, if  $F$  is unsatisfiable, there is a technique of *Ground Resolution* which gives  $\perp$  and terminates.
- ▶ However, if  $F$  is satisfiable, then this process may go on forever.
- ▶ Validity is *semi decidable* : a valid formula  $F$  has a finite witness of its validity, namely, a finite resolution refutation for  $\neg F$ .
- ▶ If  $F$  is not valid, and satisfiable, then there may not be a finite witness.
- ▶ This is for general FOL : however, we can focus on FOL over some special signatures where satisfiability is decidable.

# CS 228 : Logic in Computer Science

Krishna. S

# Satisfaction, Validity

---

- ▶ Given a FO formula  $\varphi(x_1, \dots, x_n)$  over a signature  $\tau$ , is it satisfiable/valid?
  - ▶ Satisfiable, if there exists a  $\tau$ -structure  $\mathcal{A}$  and an assignment  $\alpha$  for  $x_1, \dots, x_n$  in  $u(\mathcal{A})$  such that  $\mathcal{A} \models_\alpha \varphi(x_1, \dots, x_n)$

# Satisfaction, Validity

---

- ▶ Given a FO formula  $\varphi(x_1, \dots, x_n)$  over a signature  $\tau$ , is it satisfiable/valid?
  - ▶ Satisfiable, if there exists a  $\tau$ -structure  $\mathcal{A}$  and an assignment  $\alpha$  for  $x_1, \dots, x_n$  in  $u(\mathcal{A})$  such that  $\mathcal{A} \models_{\alpha} \varphi(x_1, \dots, x_n)$
  - ▶ Valid, if for any  $\tau$ -structure  $\mathcal{A}$  and any assignment  $\alpha$  for  $x_1, \dots, x_n$  in  $u(\mathcal{A})$ ,  $\mathcal{A} \models_{\alpha} \varphi(x_1, \dots, x_n)$
- ▶ Assume we fix the type of the structure  $\mathcal{A}$ , say words (why words?)

# FOL over Words

---

- ▶ Given an FO sentence  $\varphi$  over words, is it satisfiable/valid?
- ▶ Satisfiable (**Valid**) iff some word (**all words**) satisfies  $\varphi$

# FOL over Words

---

- ▶ Given an FO sentence  $\varphi$  over words, is it satisfiable/valid?
- ▶ Satisfiable (Valid) iff some word (all words) satisfies  $\varphi$
- ▶ There could be infinitely many words  $w$  satisfying  $\varphi$
- ▶  $L(\varphi) = \{ \text{words } w \mid w \models \varphi\}$  is called the language of  $\varphi$

# FOL over Words

---

- ▶ Given an FO sentence  $\varphi$  over words, is it satisfiable/valid?
- ▶ Satisfiable (Valid) iff some word (all words) satisfies  $\varphi$
- ▶ There could be infinitely many words  $w$  satisfying  $\varphi$
- ▶  $L(\varphi) = \{ \text{words } w \mid w \models \varphi \}$  is called the language of  $\varphi$
- ▶ Given  $\varphi$ , write an algorithm to check  $L(\varphi) = \emptyset$

## First-Order Logic over Words

# Expressiveness and Satisfiability

---

- ▶ Signature for words :  $<$ ,  $S$  and  $Q_a$  for finitely many symbols  $a$
- ▶ Given a FO formula over words, the signature is fixed

# Expressiveness and Satisfiability

---

- ▶ Signature for words :  $<$ ,  $S$  and  $Q_a$  for finitely many symbols  $a$
- ▶ Given a FO formula over words, the signature is fixed
- ▶ Expressiveness
  - ▶ Given a set of words or a language  $L$ , can you write a FO formula  $\varphi$  such that  $L(\varphi) = L$

# Expressiveness and Satisfiability

---

- ▶ Signature for words :  $<$ ,  $S$  and  $Q_a$  for finitely many symbols  $a$
- ▶ Given a FO formula over words, the signature is fixed
- ▶ Expressiveness
  - ▶ Given a set of words or a language  $L$ , can you write a FO formula  $\varphi$  such that  $L(\varphi) = L$
  - ▶ If you can, FO is expressive enough to capture your language/specification/property

# Expressiveness and Satisfiability

---

- ▶ Signature for words :  $<$ ,  $S$  and  $Q_a$  for finitely many symbols  $a$
- ▶ Given a FO formula over words, the signature is fixed
- ▶ Expressiveness
  - ▶ Given a set of words or a language  $L$ , can you write a FO formula  $\varphi$  such that  $L(\varphi) = L$
  - ▶ If you can, FO is expressive enough to capture your language/specification/property
  - ▶ If you cannot, show that FO cannot capture your property.
- ▶ Satisfiability

# Expressiveness and Satisfiability

---

- ▶ Signature for words :  $<$ ,  $S$  and  $Q_a$  for finitely many symbols  $a$
- ▶ Given a FO formula over words, the signature is fixed
- ▶ Expressiveness
  - ▶ Given a set of words or a language  $L$ , can you write a FO formula  $\varphi$  such that  $L(\varphi) = L$
  - ▶ If you can, FO is expressive enough to capture your language/specification/property
  - ▶ If you cannot, show that FO cannot capture your property.
- ▶ Satisfiability
  - ▶ Given a FO formula  $\varphi$  over words, is  $L(\varphi)$  non-empty?

## A Primer for Words

# Alphabet

---

- ▶ An alphabet  $\Sigma$  is a finite set
  - ▶  $\Sigma = \{a, b, \dots, z\}$
  - ▶  $\Sigma = \{+, \alpha, 100, B\}$
- ▶ Elements of  $\Sigma$  called letters or symbols
- ▶ A word or string over  $\Sigma$  is a finite sequence of symbols from  $\Sigma$
- ▶ If  $\Sigma = \{a, b\}$ , then *abababa* is a word of length 7
- ▶ The length of a word  $w$  is denoted  $|w|$
- ▶ There is a unique word of length 0 denoted  $\epsilon$ , called the empty word
- ▶  $|\epsilon| = 0$

# Notations for Words

---

- ▶  $aaaaa$  denoted  $a^5$
- ▶  $a^0 = \epsilon$
- ▶  $a^{n+1} = a^n.a = a.a^n$
- ▶ The set of all words over  $\Sigma$  is denoted  $\Sigma^*$ 
  - ▶  $\{a, b\}^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, \dots\}$
  - ▶  $\{a\}^* = \{\epsilon, a, aa, aaa, \dots\} = \{a^n \mid n \geq 0\}$
- ▶ By convention,  $\{\}^* = \{\epsilon\}$

# Notations for Words

---

- ▶  $\Sigma$  is a finite set
- ▶  $\Sigma^*$  is the infinite set of all finite words over alphabet  $\Sigma$
- ▶ Each  $w \in \Sigma^*$  is a finite word
  - ▶  $\{a, b\} = \{b, a\}$  but  $ab \neq ba$
  - ▶  $\{a, a, b\} = \{a, b\}$  but  $aab \neq ab$
  - ▶  $\emptyset$  is the set consisting of no words
  - ▶  $\{\epsilon\}$  is a set having the single word  $\epsilon$
  - ▶  $\epsilon$  is a word

# Operations on Words

---

- ▶ Concatenation of words :  $x.y = xy$ 
  - ▶ Concatenation is associative :  $x.(yz) = (xy).z$
  - ▶ Concatenation not commutative in general  $x.y \neq y.x$
  - ▶  $\epsilon$  is the identity for concatenation  $\epsilon.x = x.\epsilon = x$
  - ▶  $|x.y| = |x| + |y|$
- ▶  $x^n$  : catenating word  $x$   $n$  times
  - ▶  $(aab)^5 = \textcolor{red}{aab} \textcolor{blue}{aba} \textcolor{magenta}{aab} \textcolor{green}{aba} \textcolor{blue}{aab}$
  - ▶  $(aab)^0 = \epsilon$
  - ▶  $(aab)^* = \{\epsilon, aab, aabaab, aabaabaab, \dots\}$
  - ▶  $x^{n+1} = x^n x$

# More Operations on Words

---

- ▶ For  $a \in \Sigma$  and  $x \in \Sigma^*$ ,

$|x|_a = \text{number of times the symbol } a \text{ occurs in the word } x$

- ▶  $|aabbaa|_a = 4, |aabbaa|_b = 2$
- ▶  $|\epsilon|_a = 0$
- ▶ Prefix of a word  $w \in \Sigma^*$  is an initial subword of  $w$

$$\text{Pref}(w) = \{x \in \Sigma^* \mid \exists y \in \Sigma^*, w = x.y\}$$

- ▶  $\text{Pref}(aaba} = \{\epsilon, a, aa, aab, aaba\}$
- ▶ Proper prefixes =  $\{a, aa, aab\}$
- ▶  $\epsilon, aaba$  improper prefixes

# Operation on Sets

---

Given a finite alphabet  $\Sigma$ , denote by  $A, B, C, \dots$  subsets of  $\Sigma^*$

- ▶ Subsets of  $\Sigma^*$  are called languages
- ▶  $A \cup B = \{x \in \Sigma^* \mid x \in A \text{ or } x \in B\}$ 
  - ▶  $A = a^*, B = \{b, bb\}, A \cup B = a^* \cup \{b, bb\}$
- ▶  $A \cap B = \{x \in \Sigma^* \mid x \in A \text{ and } x \in B\}$ 
  - ▶  $A = (ab)^*, B = \{abab, \epsilon, bb\}, A \cap B = \{\epsilon, abab\}$
- ▶  $\overline{A} = \{x \in \Sigma^* \mid x \notin A\}$ 
  - ▶ For  $\Sigma = \{a\}$  and  $A = (aa)^*$ ,  $\overline{A} = \{a, a^3, a^5, \dots\}$
- ▶  $AB = \{xy \mid x \in A, y \in B\}$ 
  - ▶  $A = \{a, ba\}, B = \{\epsilon, aa, bb\}$
  - ▶  $AB = \{a, a^3, abb, ba, ba^3, babb\}$
  - ▶  $BA = \{a, ba, a^3, aaba, bba, bbba\}$

# Operation on Sets

---

For a set  $A \subseteq \Sigma^*$ ,

- ▶  $A^0 = \{\epsilon\}$
- ▶  $A^{n+1} = A \cdot A^n$ 
  - ▶  $\{a, ab\}^2 = \{a, ab\} \cdot \{a, ab\} = \{aa, aab, aba, abab\}$
  - ▶  $\{a, b\}^n = \{x \in \{a, b\}^* \mid |x| = n\}$
- ▶  $A^* = A^0 \cup A \cup A^2 \cup \dots = \bigcup_{i \geq 0} A^i$
- ▶  $A^+ = AA^* = A \cup A^2 \cup \dots = \bigcup_{i \geq 1} A^i$
- ▶ Union : Associative, commutative
- ▶ Concatenation : Associative, Non commutative
- ▶  $A \cup \emptyset = \emptyset \cup A = A$
- ▶  $\{\epsilon\}A = A\{\epsilon\} = A$
- ▶  $\emptyset A = A\emptyset = \emptyset$

# Operation on Sets

---

- ▶ Union, Intersection distribute over union
  - ▶  $A \cup (B \cap C) = (A \cup B) \cap (A \cup C)$
  - ▶  $A \cap (B \cup C) = (A \cap B) \cup (A \cap C)$
- ▶ Concatenation distributes over union
  - ▶  $A(\cup_{i \in I} B_i) = \cup_{i \in I} AB_i$
  - ▶  $(\cup_{i \in I} B_i)A = \cup_{i \in I} B_i A$
- ▶ Concatenation does not distribute over intersection
  - ▶  $A = \{a, ab\}, B = \{b\}, C = \{\epsilon\}$
  - ▶  $A(B \cap C) \neq AB \cap AC$

# FO for Languages

# Formalize in FO

---

Write FO formulae  $\varphi_i$  such that  $L(\varphi_i) = L_i$  for  $i = 1, \dots, 5$ .

- ▶  $L_1$  = Words that have exactly one occurrence of the letter  $c$
- ▶  $L_2$  = Words that begin with  $a$  and end with  $b$
- ▶  $L_3$  = Words that have no two consecutive  $a$ 's
- ▶  $L_4$  = Words in which any  $a$  is followed immediately by a  $b$
- ▶  $L_5$  = Words in which whenever an  $a$  occurs, it is followed eventually by a  $b$ , and no  $c$  occurs in between the  $a$  and the  $b$   
 $aabbabab, aabbcbccaab \in L_5, aacaab \notin L_5.$

# Satisfiability of FO over Words

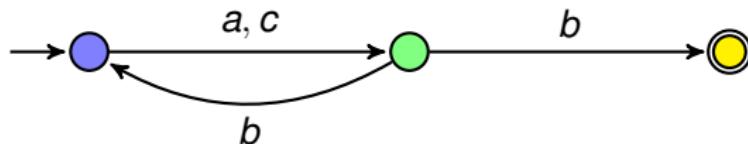
---

- ▶ Recall : Given an FO sentence  $\varphi$  over words, is  $L(\varphi) = \emptyset$ ?
- ▶ Algorithm?

# Core Idea

---

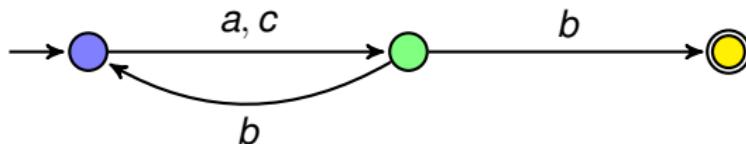
- Given FO formula  $\varphi$  over an alphabet  $\Sigma$ , construct an **edge labeled graph**  $G_\varphi$ : a graph whose edges are **labeled** by  $\Sigma$ .



# Core Idea

---

- Given FO formula  $\varphi$  over an alphabet  $\Sigma$ , construct an **edge labeled graph**  $G_\varphi$ : a graph whose edges are **labeled** by  $\Sigma$ .

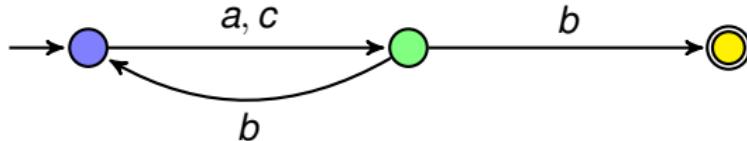


- Each path in the graph gives rise to a word over  $\Sigma$ , obtained by reading off the labels on the edges

# Core Idea

---

- Given FO formula  $\varphi$  over an alphabet  $\Sigma$ , construct an **edge labeled graph**  $G_\varphi$ : a graph whose edges are **labeled** by  $\Sigma$ .

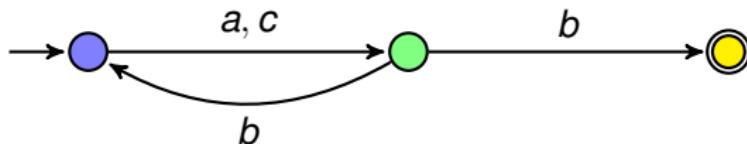


- Each path in the graph gives rise to a word over  $\Sigma$ , obtained by reading off the labels on the edges
- $G_\varphi$  has some **special** kinds of vertices
  - There is a unique vertex called the **start** vertex (**blue** vertex)
  - There are some vertices called **good** vertices (**yellow** vertex)

# Core Idea

---

- Given FO formula  $\varphi$  over an alphabet  $\Sigma$ , construct an **edge labeled graph**  $G_\varphi$ : a graph whose edges are **labeled** by  $\Sigma$ .



- Each path in the graph gives rise to a word over  $\Sigma$ , obtained by reading off the labels on the edges
- $G_\varphi$  has some **special** kinds of vertices
  - There is a unique vertex called the **start** vertex (**blue** vertex)
  - There are some vertices called **good** vertices (**yellow** vertex)
- Read off words on paths from the start vertex to any final vertex and call this set of words  $L(G_\varphi)$
- Ensure that  $G_\varphi$  is constructed such that  $L(\varphi) = L(G_\varphi)$ .

# Core Idea

---

- ▶ Why does this help?

# Core Idea

---

- ▶ Why does this help?
- ▶ We know how to check the existence of a path between 2 vertices in a graph **easily**

# Core Idea

---

- ▶ Why does this help?
- ▶ We know how to check the existence of a path between 2 vertices in a graph **easily**
- ▶ If **somewhat** we manage to construct  $G_\varphi$  **correctly**, then checking satisfiability of  $\varphi$  is same as checking the **reachability** of some good vertex from the start vertex of  $G_\varphi$ .

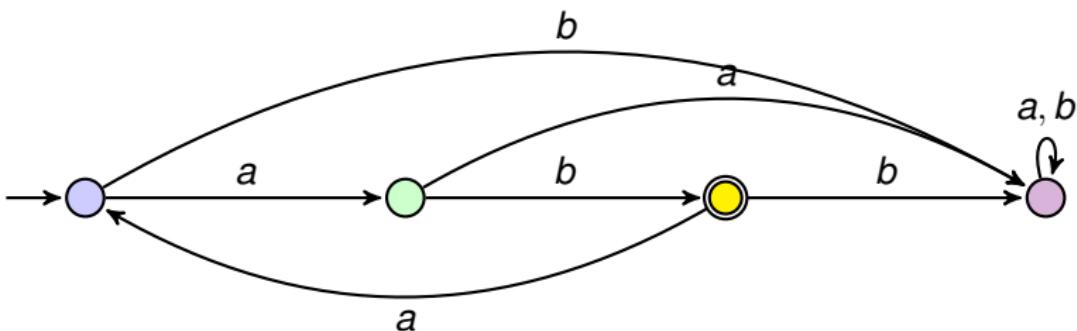
# Core Idea

---

- ▶ Why does this help?
- ▶ We know how to check the existence of a path between 2 vertices in a graph **easily**
- ▶ If **somewhat** we manage to construct  $G_\varphi$  **correctly**, then checking satisfiability of  $\varphi$  is same as checking the **reachability** of some good vertex from the start vertex of  $G_\varphi$ .
- ▶ How to construct  $G_\varphi$ ?

# A First Machine A

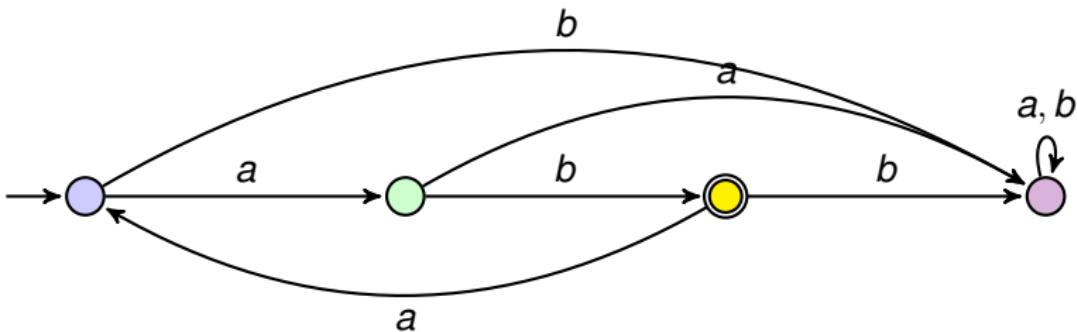
---



- ▶ Colored circles called **states**
- ▶ Arrows between circles called **transitions**
- ▶ Blue state called an **initial state**
- ▶ Doubly circled state called a **final state**

# A First Machine A

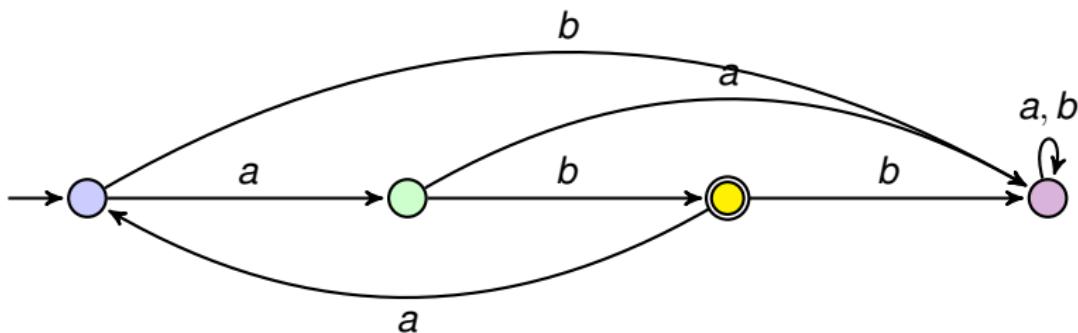
---



- ▶ A path from one state to another gives a word over  $\Sigma = \{a, b, c\}$
- ▶ The machine **accepts** words along paths from an initial state to a final state
- ▶ The set of words accepted by the machine is called the **language** accepted by the machine

# A First Machine A

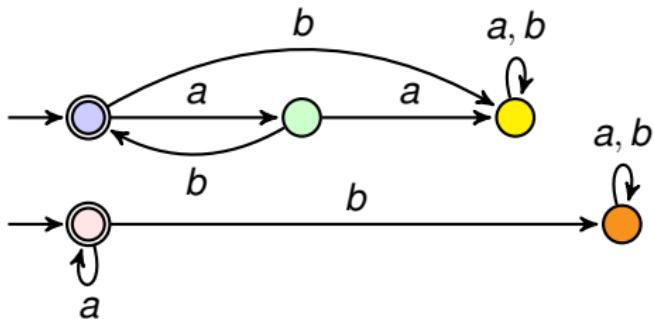
---



- ▶ What is the language  $L$  accepted by this machine,  $L(A)$ ?
- ▶ Write an FO formula  $\varphi$  such that  $L(\varphi) = L(A)$

## A Second and a Third Machine $B, C$

---



- ▶ What are  $L(B), L(C)$ ?
- ▶ Give an FO formula  $\varphi$  such that  $L(\varphi) = L(B) \cup L(C)$

# CS 228 : Logic in Computer Science

Krishna. S

# Finite State Machines

---

A deterministic finite state automaton (DFA)  $A = (Q, \Sigma, \delta, q_0, F)$

- ▶  $Q$  is a finite set of states
- ▶  $\Sigma$  is a finite alphabet
- ▶  $\delta : Q \times \Sigma \rightarrow Q$  is the transition function
- ▶  $q_0 \in Q$  is the initial state
- ▶  $F \subseteq Q$  is the set of final states
- ▶  $L(A) = \text{all words leading from } q_0 \text{ to some } f \in F$

# Languages, Machines and Logic

---

A language  $L \subseteq \Sigma^*$  is called **regular** iff there exists some DFA  $A$  such that  $L = L(A)$ .

A language  $L \subseteq \Sigma^*$  is called **FO-definable** iff there exists an FO formula  $\varphi$  such that  $L = L(\varphi)$ .

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Begins with  $a$ , ends with  $b$ , and has a pair of consecutive  $a$ 's
- ▶ Contains a  $b$  and ends with  $aa$
- ▶ Contains  $abb$
- ▶ There are two occurrences of  $b$  between which only  $a$ 's occur
- ▶ Right before the last position is an  $a$
- ▶ Even length words

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

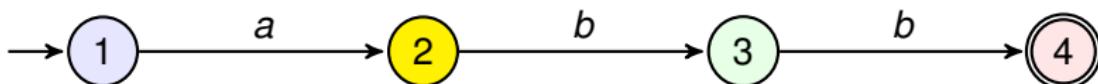
- Contains *abb*

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- Contains *abb*



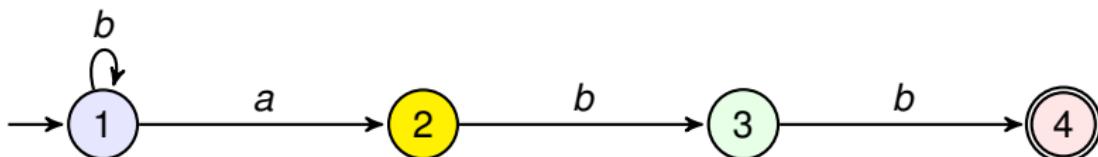
$$\exists x \exists y \exists z (Q_a(x) \wedge Q_b(y) \wedge Q_b(z) \wedge S(x, y) \wedge S(y, z))$$

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- Contains *abb*

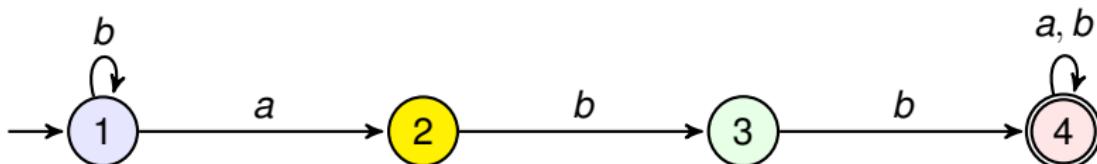


$$\exists x \exists y \exists z (Q_a(x) \wedge Q_b(y) \wedge Q_b(z) \wedge S(x, y) \wedge S(y, z))$$

# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- Contains *abb*

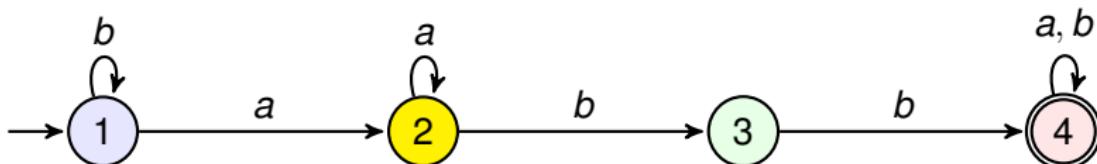


$$\exists x \exists y \exists z (Q_a(x) \wedge Q_b(y) \wedge Q_b(z) \wedge S(x, y) \wedge S(y, z))$$

# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- Contains *abb*



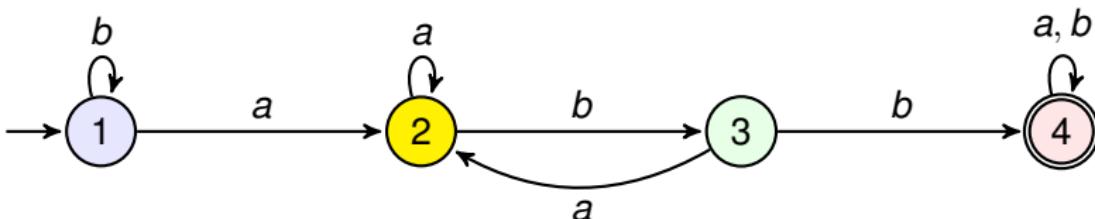
$$\exists x \exists y \exists z (Q_a(x) \wedge Q_b(y) \wedge Q_b(z) \wedge S(x, y) \wedge S(y, z))$$

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

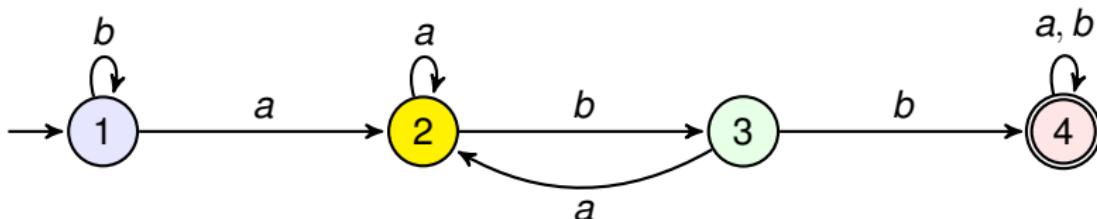
- Contains *abb*



# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- Contains  $abb$



$$\exists x \exists y \exists z (Q_a(x) \wedge Q_b(y) \wedge Q_b(z) \wedge S(x, y) \wedge S(y, z))$$

# Is it Regular? Is it FO-definable?

---

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

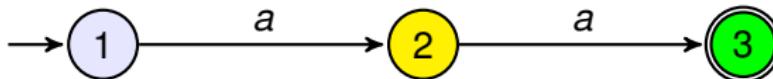
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



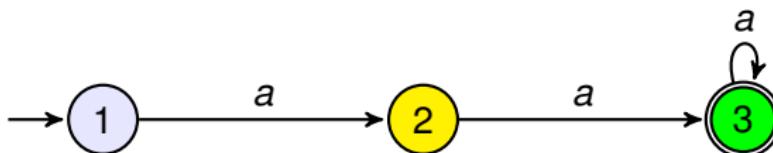
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



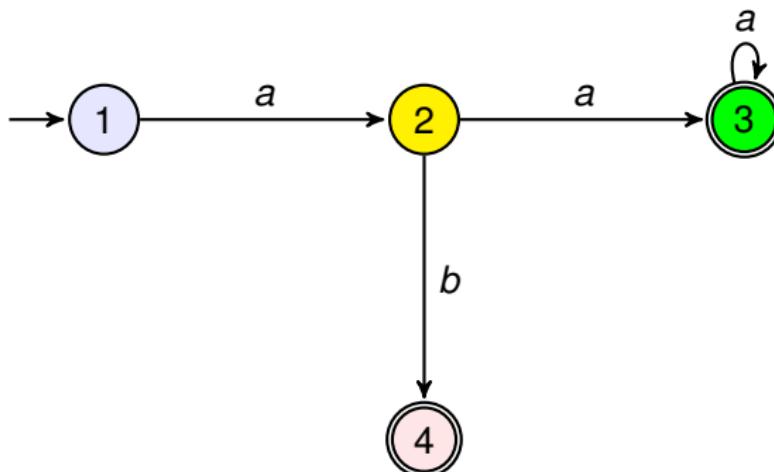
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



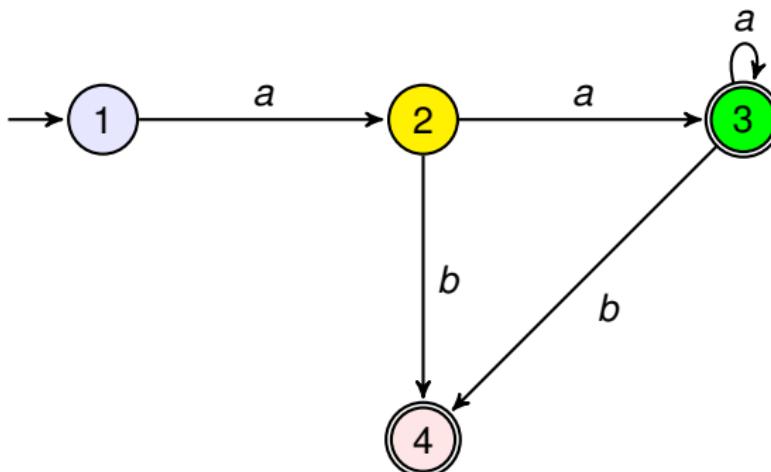
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



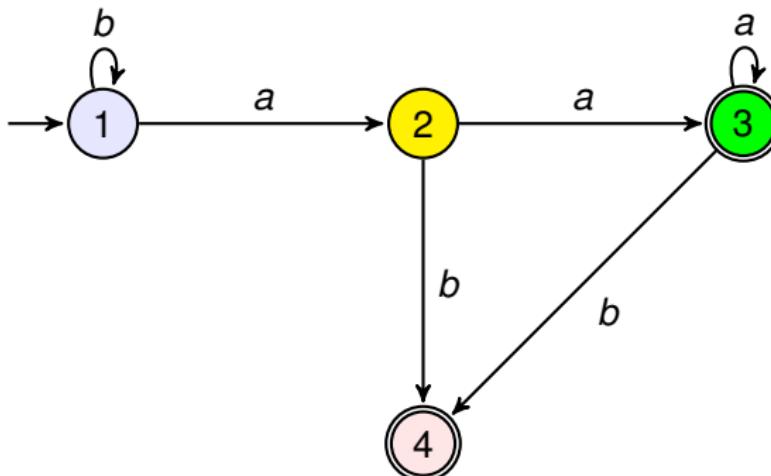
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



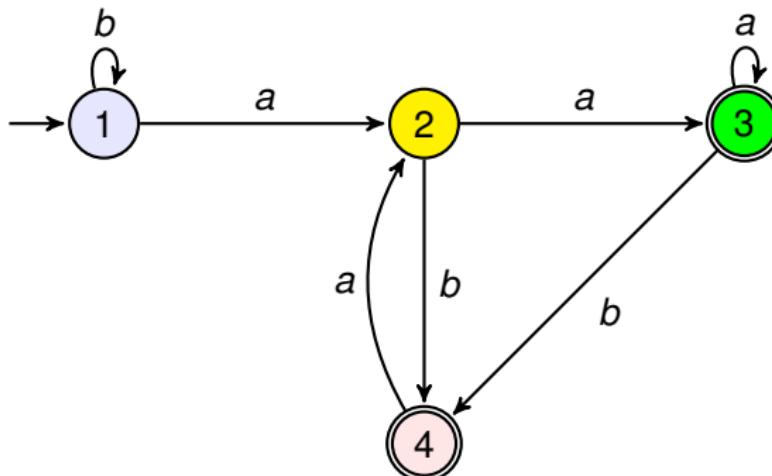
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



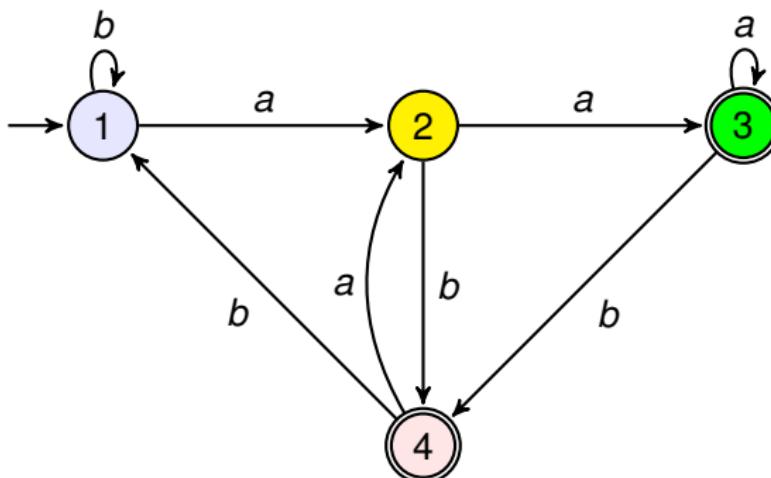
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



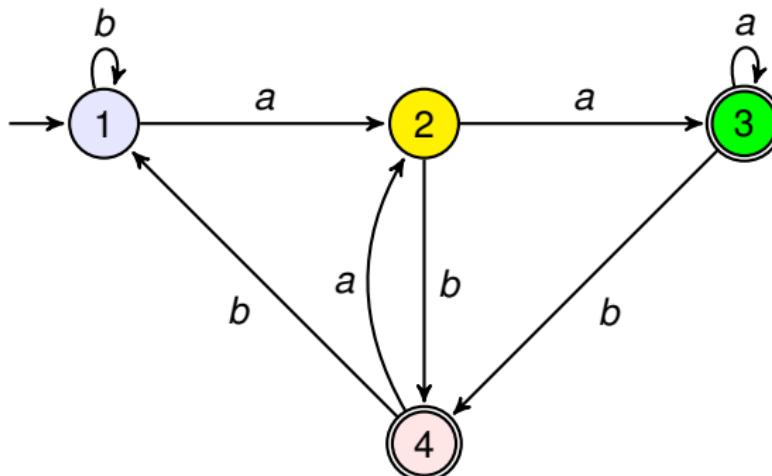
# Is it Regular? Is it FO-definable?

$\Sigma = \{a, b\}$ . Consider the following languages  $L \subseteq \Sigma^*$ :

- ▶ Right before the last position is an  $a$  :

Examples :  $ab, babbaa, bbab$

Non examples :  $ba, bb, aba$



# Deterministic Finite Automata

---

- ▶ Every state on every symbol goes to a unique state
  - ▶  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- ▶ Given a string  $w \in \Sigma^*$  and a state  $q \in Q$ , iteratively apply  $\delta$ 
  - ▶  $w = aab$
  - ▶  $\delta(q, a) = q_1,$

# Deterministic Finite Automata

---

- ▶ Every state on every symbol goes to a unique state
  - ▶  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- ▶ Given a string  $w \in \Sigma^*$  and a state  $q \in Q$ , iteratively apply  $\delta$ 
  - ▶  $w = aab$
  - ▶  $\delta(q, a) = q_1, \delta(\delta(q, a), a) = \delta(q_1, a) = q_2,$

# Deterministic Finite Automata

---

- ▶ Every state on every symbol goes to a unique state
  - ▶  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- ▶ Given a string  $w \in \Sigma^*$  and a state  $q \in Q$ , iteratively apply  $\delta$ 
  - ▶  $w = aab$
  - ▶  $\delta(q, a) = q_1, \delta(\delta(q, a), a) = \delta(q_1, a) = q_2,$   
 $\delta(\delta(\delta(q, a), a), b) =$

# Deterministic Finite Automata

---

- ▶ Every state on every symbol goes to a unique state
  - ▶  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- ▶ Given a string  $w \in \Sigma^*$  and a state  $q \in Q$ , iteratively apply  $\delta$ 
  - ▶  $w = aab$
  - ▶  $\delta(q, a) = q_1, \delta(\delta(q, a), a) = \delta(q_1, a) = q_2,$   
 $\delta(\delta(\delta(q, a), a), b) = \delta(\delta(q_1, a), b) =$

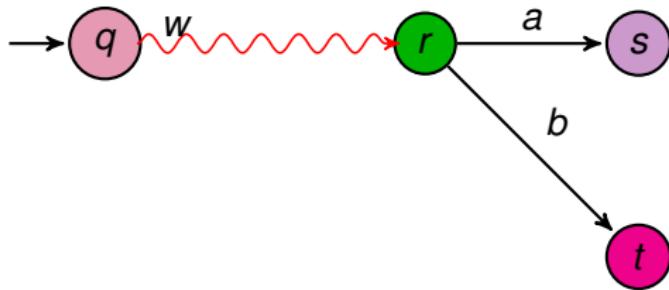
# Deterministic Finite Automata

---

- ▶ Every state on every symbol goes to a unique state
  - ▶  $\delta : Q \times \Sigma \rightarrow Q$  is a transition function
- ▶ Given a string  $w \in \Sigma^*$  and a state  $q \in Q$ , iteratively apply  $\delta$ 
  - ▶  $w = aab$
  - ▶  $\delta(q, a) = q_1, \delta(\delta(q, a), a) = \delta(q_1, a) = q_2,$   
 $\delta(\delta(\delta(q, a), a), b) = \delta(\delta(q_1, a), b) = \delta(q_2, b) = q_3$
  - ▶  $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$  extension of  $\delta$  to strings
    - ▶  $\hat{\delta}(q, \epsilon) = q$
    - ▶  $\hat{\delta}(q, wa) = \delta(\hat{\delta}(q, w), a)$

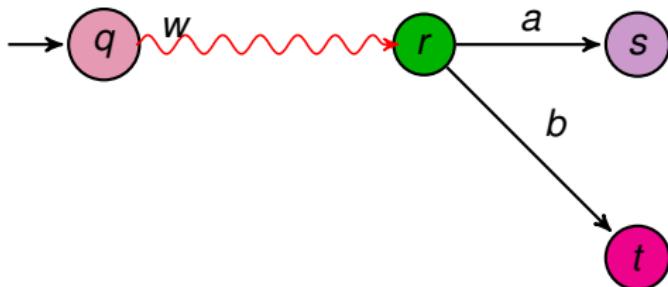
# DFA : Transition Function on Words

---



# DFA : Transition Function on Words

---



- ▶  $\hat{\delta}(q, wa) = s = \delta(\hat{\delta}(q, w), a) = \delta(r, a)$
- ▶  $\hat{\delta}(q, wb) = t = \delta(\hat{\delta}(q, w), b) = \delta(r, b)$

# DFA Acceptance

---

- ▶  $w \in \Sigma^*$  is accepted iff  $\hat{\delta}(q_0, w) \in F$
- ▶  $w \in \Sigma^*$  is rejected iff  $\hat{\delta}(q_0, w) \notin F$
- ▶ Any string  $w \in \Sigma^*$  is either accepted or rejected by a DFA  $A$
- ▶  $L(A) = \{w \in \Sigma^* \mid \hat{\delta}(q_0, w) \in F\}$
- ▶  $\Sigma^* = L(A) \cup \overline{L(A)}$

# DFA States

---

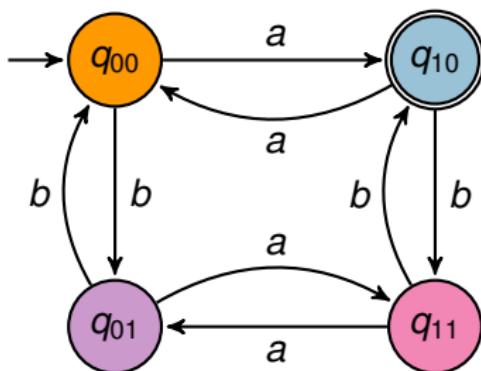
- ▶ Each state is a **bucket** holding infinitely many words
- ▶ Thus we have good and bad buckets
- ▶ The buckets partition  $\Sigma^*$
- ▶ **Good buckets** determine the language accepted by the DFA
- ▶ Words that land in bad buckets are not accepted by the DFA

# CS 228 : Logic in Computer Science

Krishna. S

# Language Acceptance : Proof

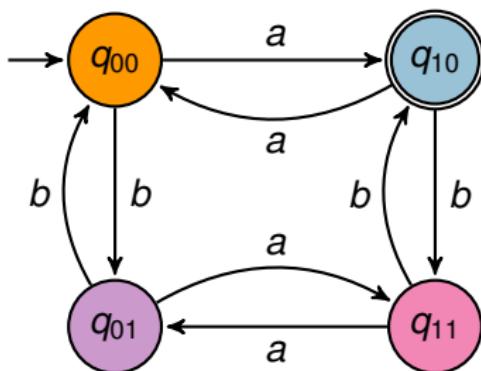
---



- ▶ Prove by induction on  $|w|$

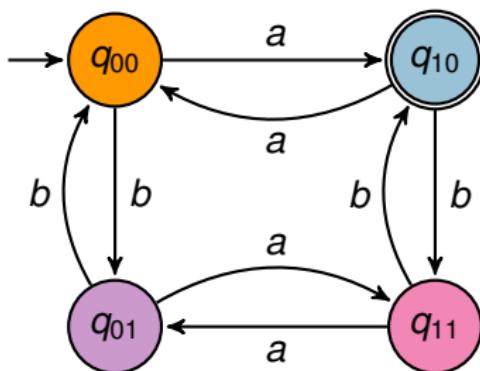
# Language Acceptance : Proof

---



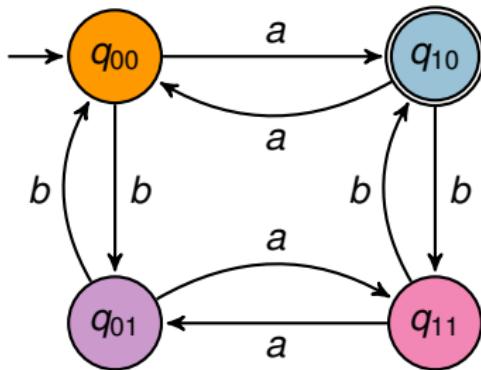
- ▶ Prove by induction on  $|w|$
- ▶ Base case : For  $|w| = \epsilon$ ,  $\hat{\delta}(q_{00}, \epsilon) = q_{00}$

# Language Acceptance : Proof



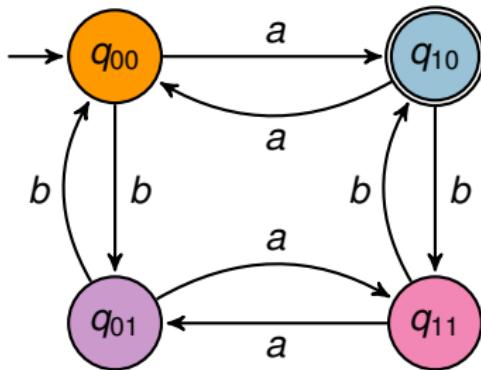
- ▶ Prove by induction on  $|w|$
- ▶ Base case : For  $|w| = \epsilon$ ,  $\hat{\delta}(q_{00}, \epsilon) = q_{00}$
- ▶ Assume the claim for  $x \in \Sigma^*$ , and show it for  $xc, c \in \{a, b\}$ .

# Language Acceptance : Proof



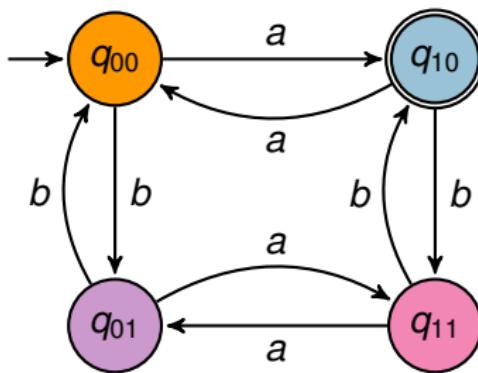
- ▶  $\hat{\delta}(q_{00}, xc) = \delta(\hat{\delta}(q_{00}, x), c)$

# Language Acceptance : Proof



- ▶  $\hat{\delta}(q_{00}, xc) = \delta(\hat{\delta}(q_{00}, x), c)$
- ▶ By induction hypothesis,  $\hat{\delta}(q_{00}, x) = q_{ij}$  iff
  - ▶ parity of  $i$  and  $|x|_a$  are the same
  - ▶ parity of  $j$  and  $|x|_b$  are the same

# Language Acceptance : Proof



- ▶ Case Analysis : If  $|x|_a$  odd and  $|x|_b$  even, then  $i = 1, j = 0$ 
  - ▶  $\delta(q_{10}, a) = q_{00}, \delta(q_{10}, b) = q_{11}$
  - ▶  $|xa|_a$  is even and  $|xa|_b$  is even
  - ▶  $|xb|_a$  is odd and  $|xb|_b$  is odd
- ▶ Other Cases : Similar
- ▶  $\hat{\delta}(q_{00}, x) = q_{10}$  iff  $|x|_a$  odd and  $|x|_b$  even

## Closure Properties : DFA

# Closure under Complementation

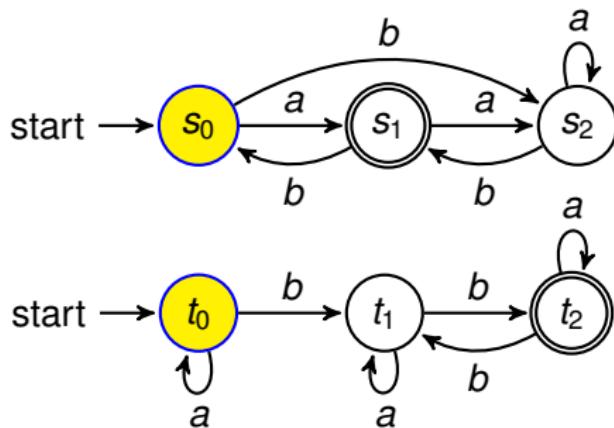
---

- ▶ If  $L$  is regular, so is  $\bar{L}$ 
  - ▶ Let  $A = (Q, q_0, \Sigma, \delta, F)$  be the DFA such that  $L = L(A)$
  - ▶ For every  $w \in L$ ,  $\hat{\delta}(q_0, w) = f$  for some  $f \in F$
  - ▶ For every  $w \notin L$ ,  $\hat{\delta}(q_0, w) = q$  for some  $q \notin F$
  - ▶ Construct  $\bar{A} = (Q, q_0, \Sigma, \delta, Q - F)$ 
    - ▶  $w \in L(\bar{A})$  iff  $\hat{\delta}(q_0, w) \in Q - F$  iff  $w \notin L(A)$
    - ▶  $L(\bar{A}) = \overline{L(A)}$

# Closure under Intersection

---

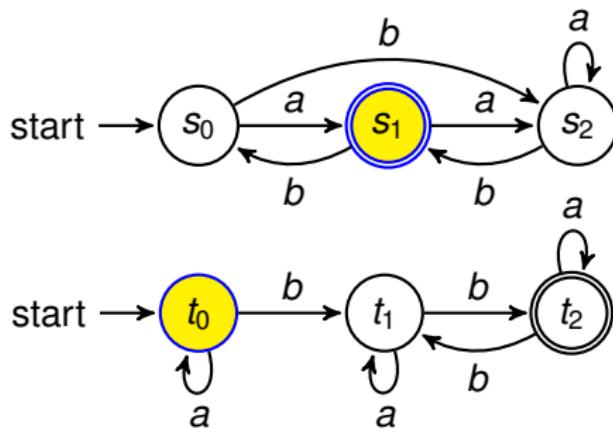
▶  $aaab$



# Closure under Intersection

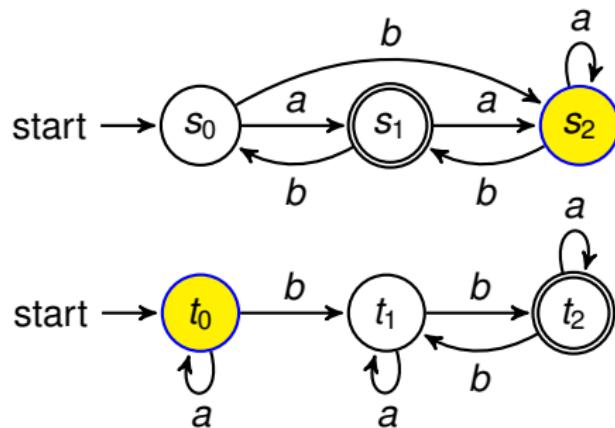
---

►  $aabb$



# Closure under Intersection

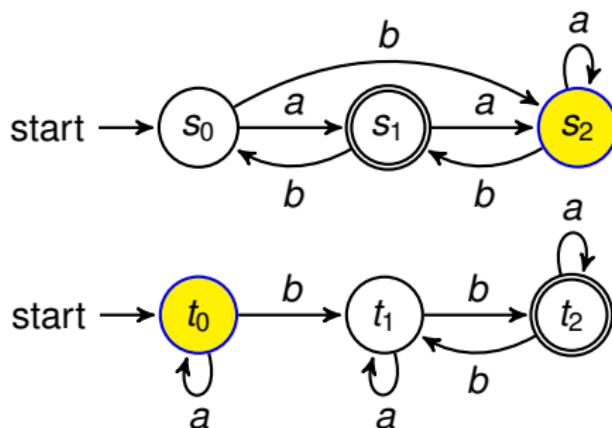
- ▶  $a\textcolor{red}{a}ab$



# Closure under Intersection

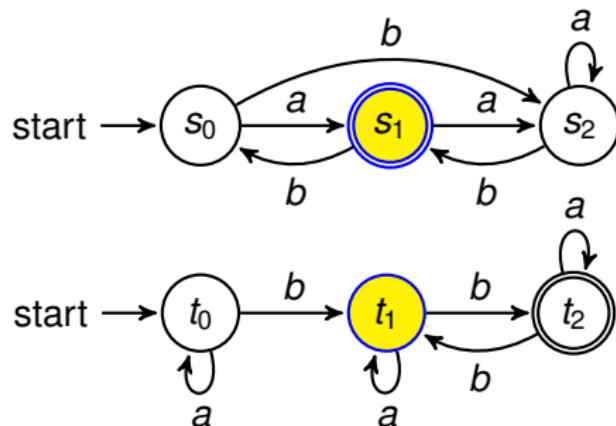
---

- ▶  $aa\textcolor{red}{ab}$



# Closure under Intersection

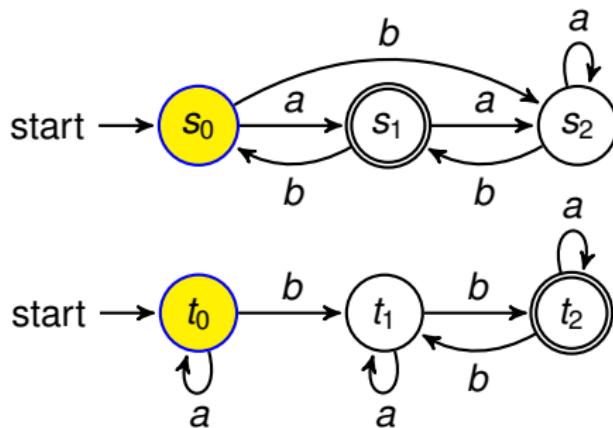
- ▶  $aaa\textcolor{red}{b}$



# Closure under Intersection

---

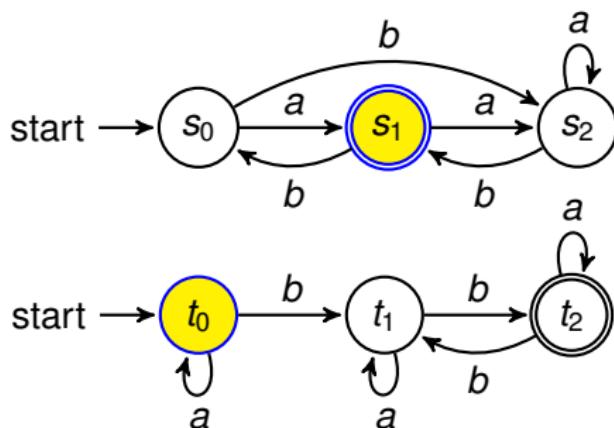
▶  $aabba$



# Closure under Intersection

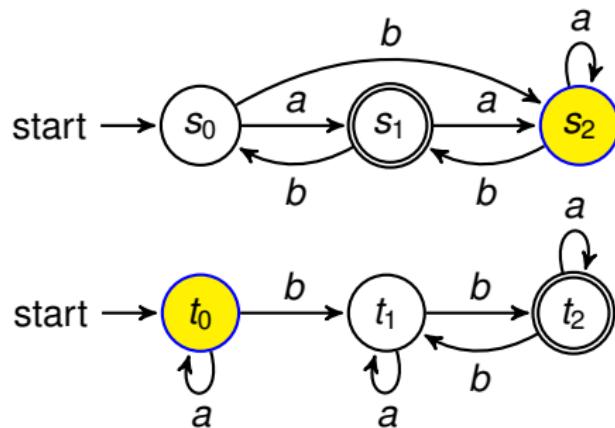
---

► *aabba*



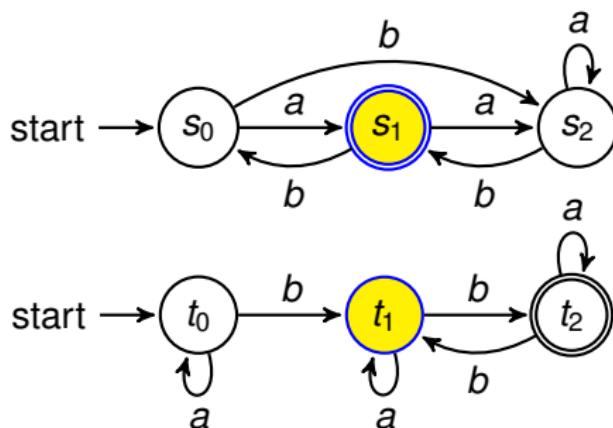
# Closure under Intersection

- ▶  $a\textcolor{red}{abba}$



# Closure under Intersection

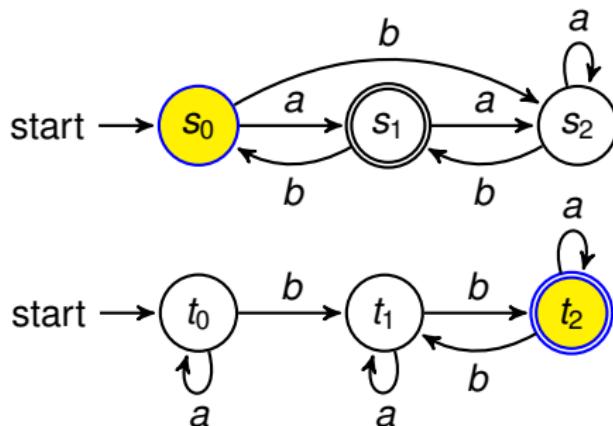
- ▶  $aabbba$



# Closure under Intersection

---

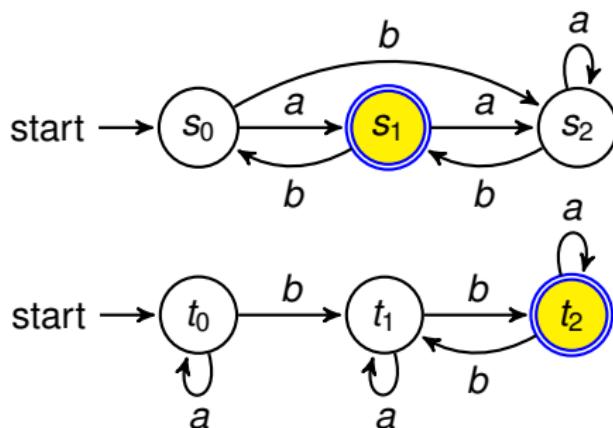
►  $aabba$



# Closure under Intersection

---

►  $aabbba$



# Closure under Intersection

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
- ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
- ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
  - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
  - ▶  $F = F_1 \times F_2$

# Closure under Intersection

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
- ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
- ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
  - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
  - ▶  $F = F_1 \times F_2$
- ▶ Show that for all  $x \in \Sigma^*$ ,  $\hat{\delta}((p, q), x) = (\hat{\delta}_1(p, x), \hat{\delta}_2(q, x))$
- ▶  $x \in L(A)$  iff  $\hat{\delta}((q_0, s_0), x) \in F$

# Closure under Intersection

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
  - ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
  - ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
    - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
    - ▶  $F = F_1 \times F_2$
  - ▶ Show that for all  $x \in \Sigma^*$ ,  $\hat{\delta}((p, q), x) = (\hat{\delta}_1(p, x), \hat{\delta}_2(q, x))$
- $x \in L(A)$  iff  $\hat{\delta}((q_0, s_0), x) \in F$  iff  $(\hat{\delta}_1(q_0, x), \hat{\delta}_2(s_0, x)) \in F_1 \times F_2$

# Closure under Intersection

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
  - ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
  - ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
    - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
    - ▶  $F = F_1 \times F_2$
  - ▶ Show that for all  $x \in \Sigma^*$ ,  $\hat{\delta}((p, q), x) = (\hat{\delta}_1(p, x), \hat{\delta}_2(q, x))$
- $x \in L(A)$  iff  $\hat{\delta}((q_0, s_0), x) \in F$  iff  $(\hat{\delta}_1(q_0, x), \hat{\delta}_2(s_0, x)) \in F_1 \times F_2$  iff  
 $\hat{\delta}_1(q_0, x) \in F_1$  and  $\hat{\delta}_2(s_0, x) \in F_2$

# Closure under Intersection

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
  - ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
  - ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
    - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
    - ▶  $F = F_1 \times F_2$
  - ▶ Show that for all  $x \in \Sigma^*$ ,  $\hat{\delta}((p, q), x) = (\hat{\delta}_1(p, x), \hat{\delta}_2(q, x))$
- $x \in L(A)$  iff  $\hat{\delta}((q_0, s_0), x) \in F$  iff  $(\hat{\delta}_1(q_0, x), \hat{\delta}_2(s_0, x)) \in F_1 \times F_2$  iff  
 $\hat{\delta}_1(q_0, x) \in F_1$  and  $\hat{\delta}_2(s_0, x) \in F_2$  iff  $x \in L(A_1)$  and  $x \in L(A_2)$

# Closure under Union

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
- ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
- ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F)$ ,
  - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$

# Closure under Union

---

- ▶  $A_1 = (Q_1, \Sigma, \delta_1, q_0, F_1)$
- ▶  $A_2 = (Q_2, \Sigma, \delta_2, s_0, F_2)$
- ▶  $A = (Q_1 \times Q_2, \Sigma, \delta, (q_0, s_0), F),$ 
  - ▶  $\delta((q, s), a) = (\delta_1(q, a), \delta_2(s, a))$
  - ▶  $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$
- ▶ Show that for all  $x \in \Sigma^*$ ,  $\hat{\delta}((p, q), x) = (\hat{\delta}_1(p, x), \hat{\delta}_2(q, x))$

$x \in L(A)$  iff  $x \in L(A_1)$  or  $x \in L(A_2)$

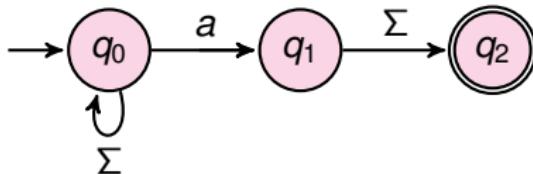
# Moving on to Non-determinism

---

- ▶ We looked at DFA
- ▶ Showed closure under union, intersection and complementation
- ▶ Before we examine closure under concatenation, we look at a more relaxed model, which is as good as a DFA

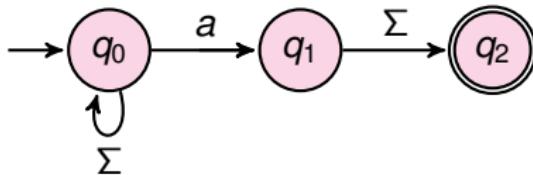
# The Comfort of Non-determinism

---



# The Comfort of Non-determinism

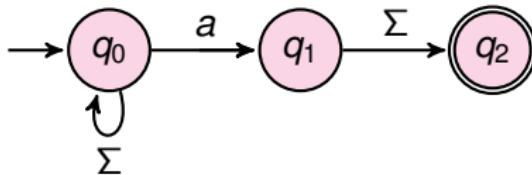
---



- ▶ Assume we relax the condition on transitions, and allow
  - ▶  $\delta : Q \times \Sigma \rightarrow 2^Q$
  - ▶  $\delta(q_0, a) = \{q_0, q_1\}, \delta(q_2, a) = \emptyset$

# The Comfort of Non-determinism

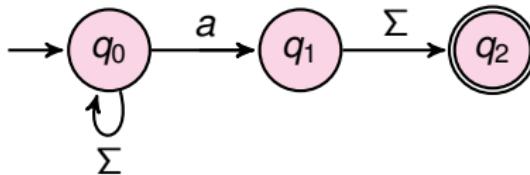
---



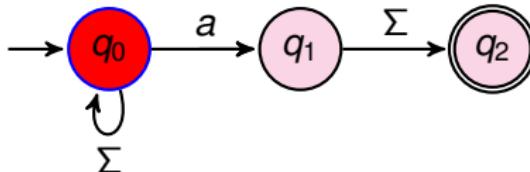
- ▶ Assume we relax the condition on transitions, and allow
  - ▶  $\delta : Q \times \Sigma \rightarrow 2^Q$
  - ▶  $\delta(q_0, a) = \{q_0, q_1\}, \delta(q_2, a) = \emptyset$
  - ▶ Is  $aabb$  accepted?

# The Comfort of Non-determinism

---



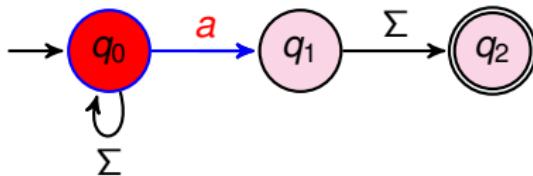
- ▶ Assume we relax the condition on transitions, and allow
  - ▶  $\delta : Q \times \Sigma \rightarrow 2^Q$
  - ▶  $\delta(q_0, a) = \{q_0, q_1\}, \delta(q_2, a) = \emptyset$
  - ▶ Is  $aabb$  accepted?



# One run of $aabb$

---

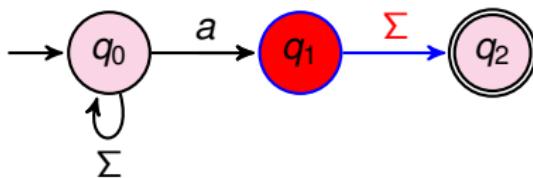
Is  $aabb$  accepted?



# One run of $aabb$

---

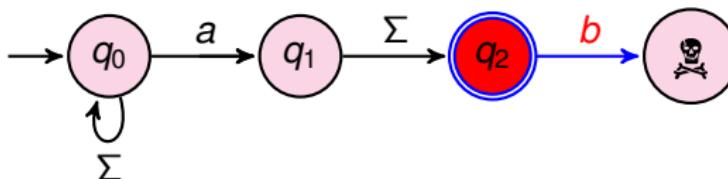
Is  $aabb$  accepted?



# One run of $aabb$

---

Is  $aabb$  accepted?

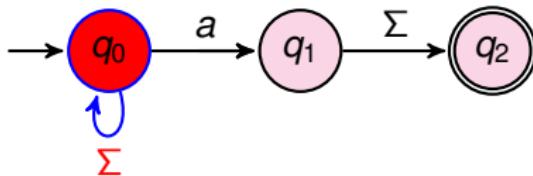


- ▶ A non-accepting run for  $aabb$

# Another run of $aabb$

---

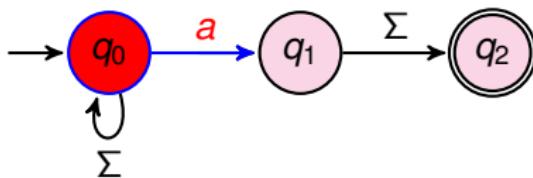
Is  $aabb$  accepted?



# Another run of $aabb$

---

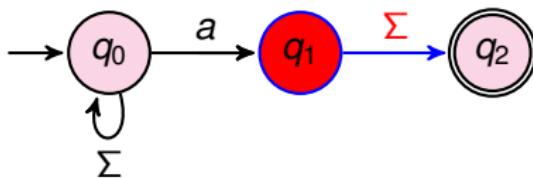
Is  $aabb$  accepted?



# Another run of *aabb*

---

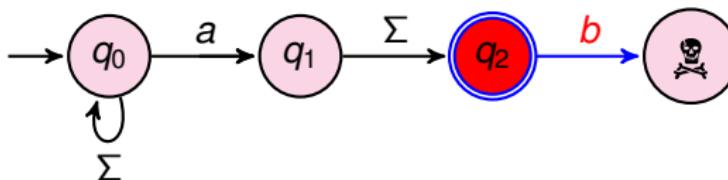
Is *aabb* accepted?



# Another run of $aabb$

---

Is  $aabb$  accepted?

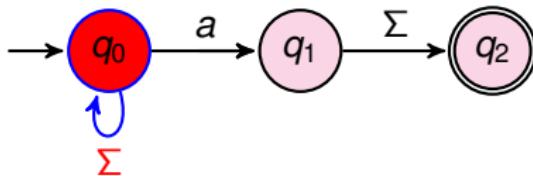


- ▶ A non-accepting run for  $aabb$

# A run of $aaab$

---

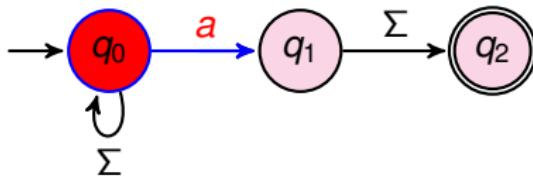
Is  $aaab$  accepted?



# A run of $aaab$

---

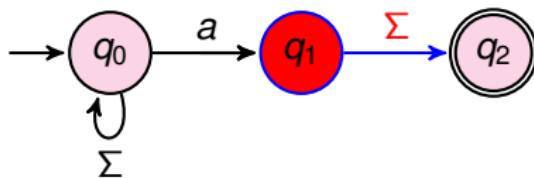
Is  $aaab$  accepted?



# A run of $aaab$

---

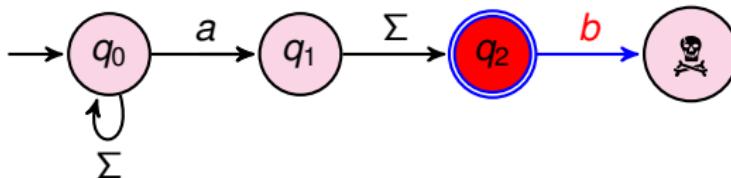
Is  $aaab$  accepted?



# A run of $aaab$

---

Is  $aaab$  accepted?

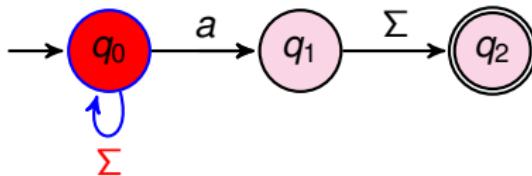


- ▶ A non-accepting run for  $aaab$

# Another run of *aaab*

---

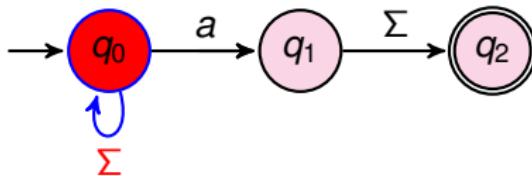
Is *aaab* accepted?



# Another run of $aaab$

---

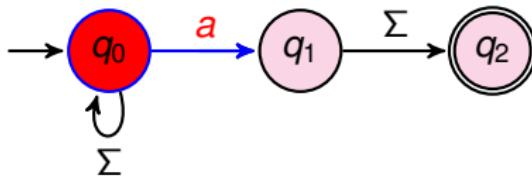
Is  $aaab$  accepted?



# Another run of $aaab$

---

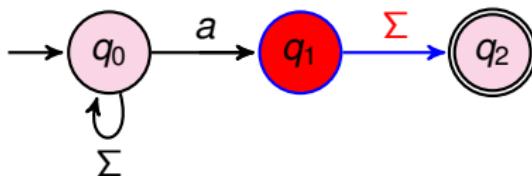
Is  $aa\textcolor{red}{a}b$  accepted?



# Another run of *aaab*

---

Is *aaab* accepted?



- ▶ An accepting run for *aaab*

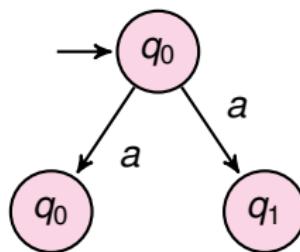
# Nondeterministic Finite Automata(NFA)

---

- ▶  $N = (Q, \Sigma, \delta, Q_0, F)$ 
  - ▶  $Q$  is a finite set of states
  - ▶  $Q_0 \subseteq Q$  is the set of initial states
  - ▶  $\delta : Q \times \Sigma \rightarrow 2^Q$  is the transition function
  - ▶  $F \subseteq Q$  is the set of final states
- ▶ Acceptance condition : A word  $w$  is accepted iff it has atleast one accepting path

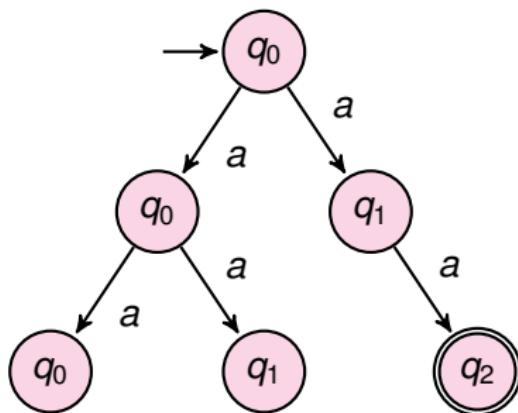
## Run Tree of aaab

---



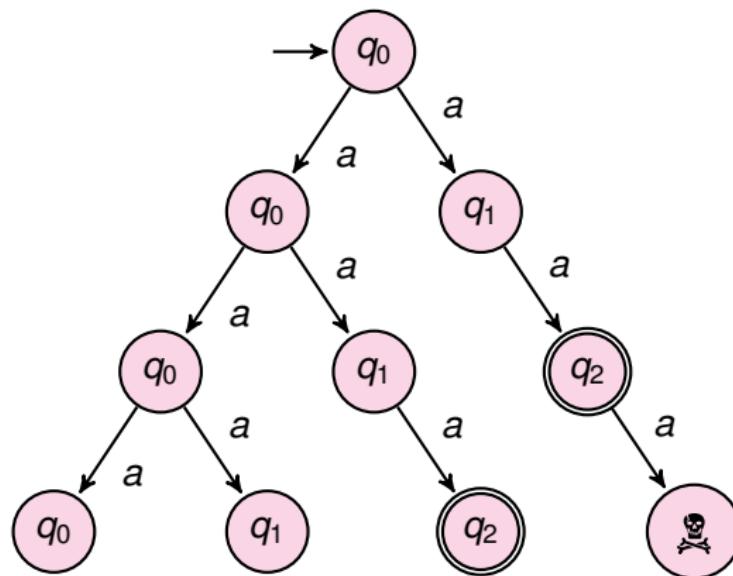
## Run Tree of aaab

---



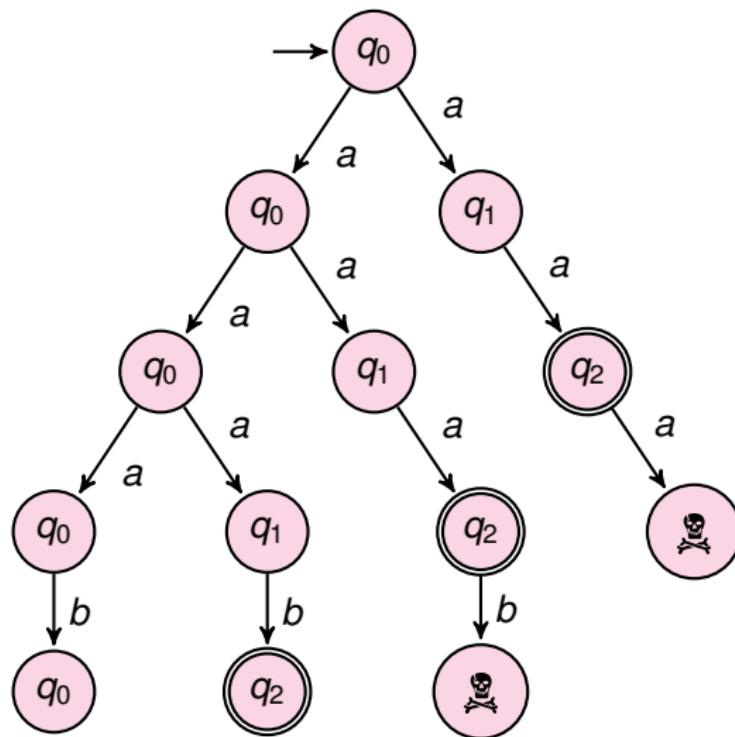
# Run Tree of aaab

---



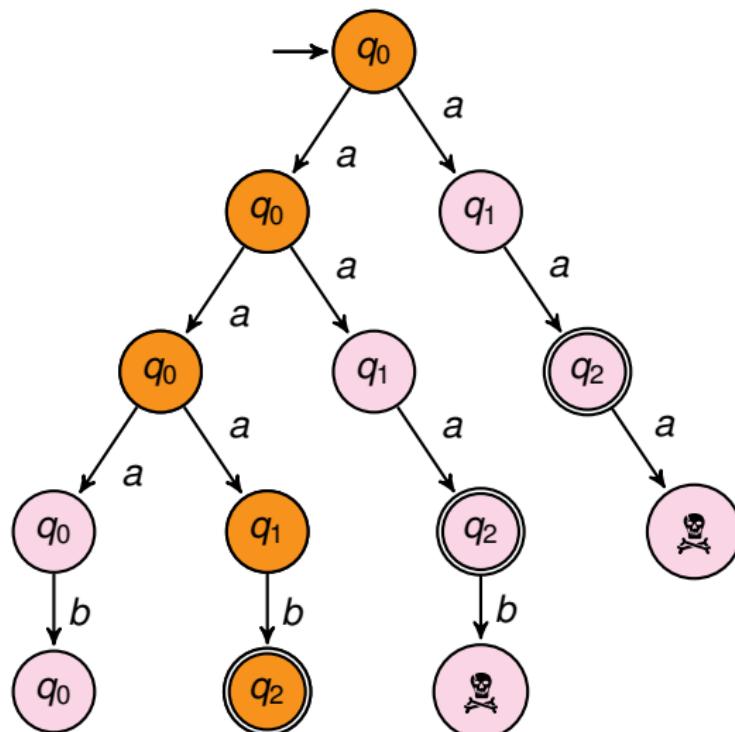
# Run Tree of aaab

---



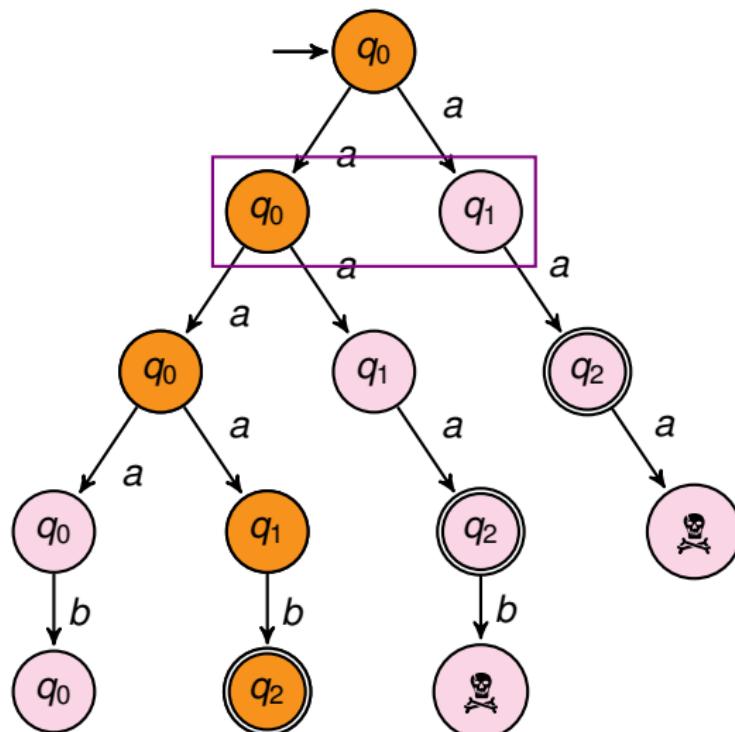
# Run Tree of aaab

---

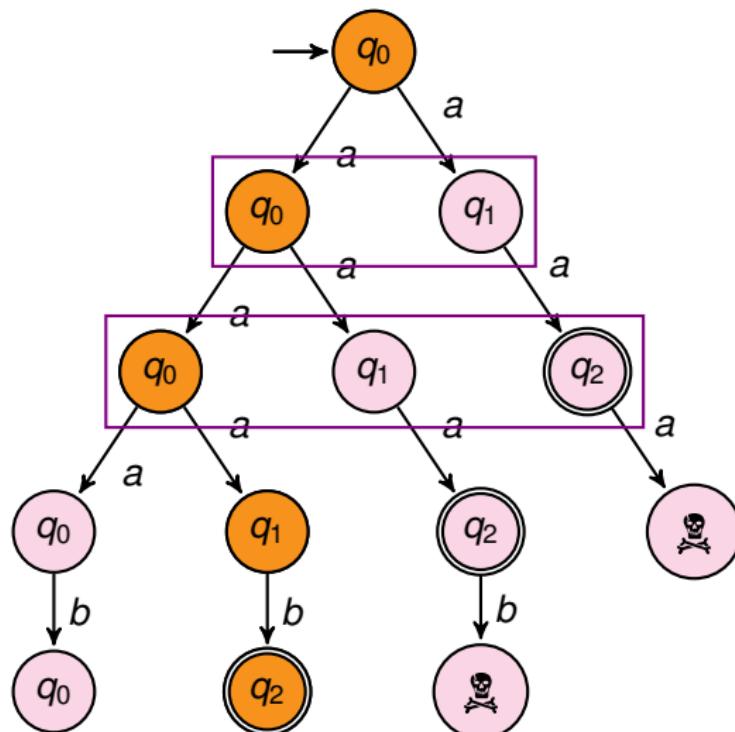


# Run Tree of aaab

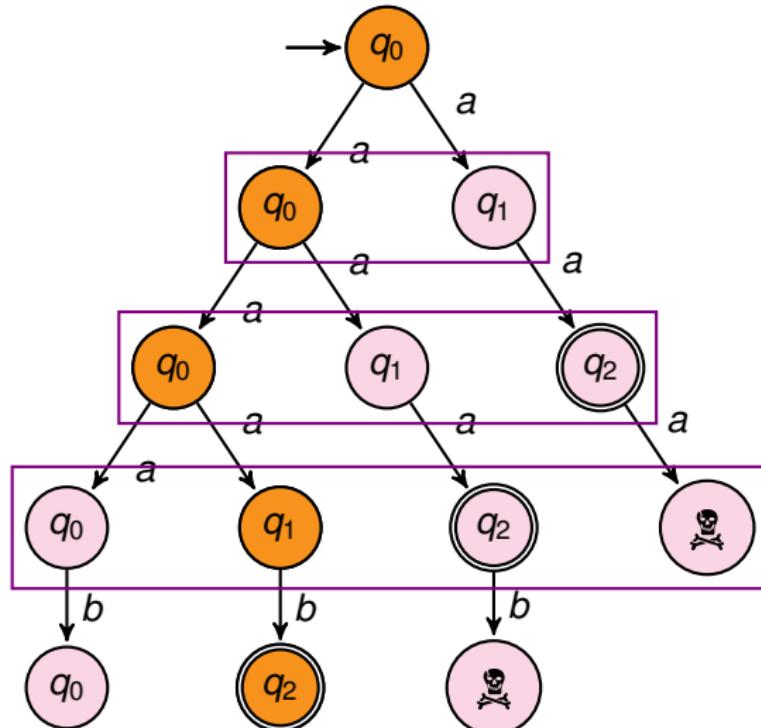
---



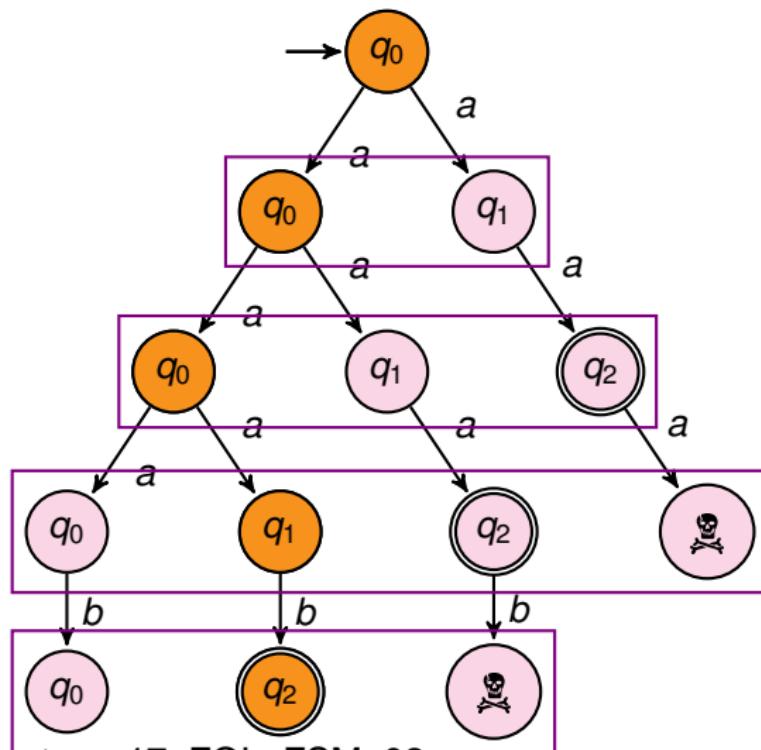
# Run Tree of aaab



# Run Tree of aaab



# Run Tree of aaab



# The Single Run

---

