BITS Pilani Hyderabad Campus
CS F214 Logic in Computer Science,
I Semester 2021-2022
Lecture Notes
Lecture 1-5

1 Barber Paradox

There is a lone male barber who lives on an island. The barber shaves all those men who do not shave themselves and only those men.

Consider following statement and determine if it is true or false? Statement: The barber shaves himself.

Such statements are called as propositions. (Formal definition will be discussed in later lectures).

This statement is paradoxical. The barber cannot shave himself as he only shaves those who do not shave themselves. Thus, if he shaves himself he ceases to be the barber (i.e. person who cannot shave). On the other hand, if barber does not shave himself, then he belongs to the group of people who would be shaved by the barber and hence barber should shave himself.

Hence, there is no clear answer to proposition.

2 Naive Set Theory

- This has been proposed by *Georg Cantor*.
- A SET is a collection of objects or entities.
- Bertrand Russell proposed Russell's paradox for above definition

2.1 Russell's Paradox

Let R be the set of all sets that are not members of themselves.

$$A = \{1, 2, 3, 4\}, \therefore A \in R$$
$$R = \{X | X \notin X\}$$

Is the statement $R \in R$ true or false?

Similar to the Barber Paradox, if we assume that statement is true that R now belongs to itself. Now, R is a set which contains itself and hence should not be included in R. If we assume statement is false that R does not belong to itself. Then, as per the definition R is now a set which does not contain itself and hence should be included in the set R.

Hence, answering this results in paradox. There is no clear answer to the statement.

2.2 Zermelo Frankel Set Theory

Ernst Zermelo and Abraham Frankel came up with axiomatic approach to resolve Russell's Paradox.

Example:

1. If the train arrives late and there are no taxis at the station, then John is late for the meeting.
2. John is not late for the meeting.
3. The train did arrive late.

$\therefore$ There were taxis at the station.

Solution: p = Train arrives late. q = There are no taxis at the station. r = John is late for meeting.

$$1.((p \wedge q) \rightarrow (r))$$
$$2.\neg r$$
$$3.p$$
$$4.\neg q \quad \text{(conclusion)}$$

Statement 1 is true when proposition q is false. i.e. when conclusion is true. Hence the conclusion follows.

Example:

1. If it is raining and Jane does not have umbrella then she will get wet.
2. Jane is not wet.
3. It is raining.

$\rightarrow$ Therefore, Jane has her umbrella with her.

Argument 1	Corresponding argument 2
Train is late (p)	It is raining
There are taxis at the station(q)	Jane has an umbrella
John is late (r)	Jane is wet

3 Propositional Logic

3.1 Declarative Sentences Or Propositions

Sentences that are in principle, either true or false (but not both).

Examples:

1. This is a class on logic.

2. The sun orbits the earth.

3. The sum of 2 and 2 is 5.

4. Every even natural number greater than 2 is the sum of 2 prime numbers. (Goldbach's conjecture. We are not sure whether this statement is true or false.)

3.2 Non-Declarative Sentences

1. Pay Attention!

2. Would you accompany me to class?

3. May fortune come your way.

Non-Declarative Sentences are not propositions.

3.3 Twin Prime Conjecture

- For all the numbers n, there exists an $n_0 > n$, such that n_0 and $n_0 + 2$ are prime numbers.
- Twin prime conjecture has not been proved yet.

3.4 Bounded Gap Theorem

- For all natural numbers n, there exists a constant c, for all natural numbers $n_0 > n$, so that there are two prime numbers in interval $[n_0, n_0 + c]$
- This has been proved by Yitang Zhang.

3.5 Truth Tables

Given these atomic propositions p,q,r,.. or p1,p2,p3.. we can combine them in compositional ways to form other propositions using logical connections.

p	$\neg p$
T	F
F	T

Table 1: Negation ($\neg$)

p	q	$p \wedge q$
F	F	F
F	T	F
T	F	F
T	T	T

Table 2: Conjuction ($\wedge$)

p	q	$p \vee q$
F	F	F
F	T	T
T	F	T
T	T	T

Table 3: Disjunction ($\vee$)

p	q	$p \rightarrow q$
F	F	T
F	T	T
T	F	F
T	T	T

Table 4: Conditional (Implication)

p $\leftrightarrow$ q $\implies$ (p$\rightarrow$q) $\wedge$ (q$\rightarrow$p)

4 Rules for Natural Deduction

4.1 And Introduction

- Given premises ϕ, ψ, we can conclude $\phi \wedge \psi$.

$$\frac{\phi, \psi}{\phi \wedge \psi} \wedge_i$$

4.2 And Elimination

- Given premises $\phi \wedge \psi$, we can conclude ϕ.

$$\frac{\phi \wedge \psi}{\phi} \wedge_{e_1}$$

$$\frac{\phi \wedge \psi}{\psi} \wedge_{e2}$$

Example Prove that the following statement is valid p, $p \wedge q$, $r \vdash q \wedge r$

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

4.3 Double Negation

Double Negation Elimination

$$\frac{\neg\neg\phi}{\phi}\neg\neg e$$

Double Negation Introduction

$$\frac{\neg\neg\phi}{\phi}\neg\neg i$$

4.4 Eliminating Implication (Modus Ponens)

$$\frac{\phi, \phi \to \psi}{\psi} \to e$$

Example: p : It rained. q : The street is wet. $p \to q$: If it rained then the street is wet

4.5 Modus Tollens

$$\frac{\phi \to \psi, \neg\psi}{\neg\phi} \to MT$$

e.g. $p \to q$: If it rained then the street is wet. (premise) $\neg q$: The street is not wet. (premise) $\therefore$ it did not rain ($\neg p$) (conclusion)

Prove the following statement $p \to (q \to r), p, \neg r \vdash \neg q$.

$1.p \to (q \to r)$	premise
$2.p$	premise
$3.\neg r$	premise
$4.q \to r$	$\to$e 2,1
$5.\neg q$	MT 4,3

By using Modus Tollens Elimination,

$$\frac{p, p \to (q \to r)}{q \to r}$$

By using Modus Tollens,

$$\frac{q \to r, \neg r}{\neg q}$$

4.6 Implies Introduction

- We wish to build implications that do not already appear as premises (or parts theorem) in our proof.

Example Prove if following condition is valid: $p \to q \vdash \neg q \to \neg p$

$1.p \to q$	premise
$2.\neg q$	assumption
$3.\neg p$	Modus Tollens 1,2
$4.\neg q \to \neg p$	[$\to$i 2,3]

2,3 $\implies$ Scope of the assumption.
* Implies introduction is formulated as :

$$\frac{\phi, \psi}{\phi \to \psi} \to i$$

BITS Pilani Hyderabad Campus
CS F214 Logic in Computer Science,
I Semester 2021-2022
Lecture Notes
Lecture 6-10

Prove: $\neg q \to \neg p \vdash p \to q$

Solution:

1.$\neg q \to \neg p$	premise
2.p	assumption
3.$\neg\neg p$	$\neg\neg i$ 2
4.$\neg\neg q$	MT 3,1
5.q	$\neg\neg e$ 4
6.$p \to q$	$\to$ i 2-5

Consider,

1.p	assumption
2.$p \to p$	$\to$ i 1-1

Above is a poof of the statement $\vdash (p \to p)$

Def: A logical formula ϕ with a valid sequent $\vdash \phi$ is called a ***Theorem.***
Remarks: Any sequent $\phi \vdash \psi$ is equivalent to $\vdash \phi \to \psi$.

Proof of $\vdash \phi \to \psi$:

1.ϕ	assumption
2.	
.	
.	
$n.\psi$	<>
$\phi \to \psi$	$\to i$ 1-n (n+1)

Remark(b): $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is equivalent to $\vdash \phi_1 \to (\phi_2 \to (...(.. \to (\phi_n \to \psi))$

5 Rules for Disjunction

5.1 OR-Introduction

$$\frac{\phi}{\phi \vee \psi} \vee_{i1}$$

$$\frac{\phi}{\psi \vee \phi} \vee_{i2}$$

5.2 OR-Elimination

$$\frac{\phi \vee \psi \qquad \begin{array}{|c|}\hline \phi \\ \vdots \\ \chi \\ \hline \end{array} \qquad \begin{array}{|c|}\hline \psi \\ \vdots \\ \chi \\ \hline \end{array}}{\chi} \vee_e$$

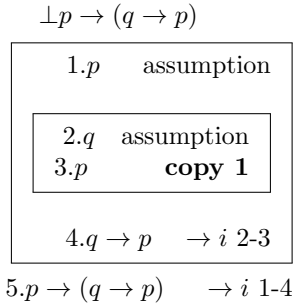
Prove: $p \vee q \vdash q \vee p$

1.	$p \vee q$	premise
	2. q	assumption
	3. $q \vee p$	$\vee_{i2}$ 2
	4. q	assumption
	5. $q \vee p$	$\vee_i 4$
6.	$q \vee p$	$\vee_e$ 1,2-3,4-5

Prove: $q \to r \vdash p \vee q \to p \vee r$

1.	$q \to r$	premise
	2. $p \vee q$	assumption
	3. p	assumption
	4. $p \vee r$	$\vee_i$ 3
	5. q	assumption
	6. r	$\to e$ 5,1
	7. $p \vee r$	$\vee_{i2}$ 6
	8. $p \vee r$	$\vee_e$ 2,3-4,5-7
9.	$(p \vee q) \to (p \vee r)$	$\to$ i 2-8

5.3 "Copy" Rule



6 Rules for Negation

6.1 Contradiction

- Contradictions are expressions of the form $\phi \wedge \neg\phi$ or $\neg\phi \wedge \phi$ where ϕ is any proposition.
- $\perp$ represents a contradiction.
- Any proposition can be derived from contradiction.

6.2 Bottom Elimination

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

6.3 Not Elimination

$$\frac{\phi \quad \neg\phi}{\perp}\neg e$$

Example: Prove $\neg p \vee q \vdash p \rightarrow q$ is valid.

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

$2.\neg p$	assumption
$3.p$	assumption
$4.\bot$	$\neg e3, 2$
$5.q$	$\bot e4$
$6.p \rightarrow q$	$[\rightarrow i3 - 5]$

$7.q$	assumption
$8.p$	assumption
$9.q$	from 7
$10.p \rightarrow q$	$[\rightarrow i8 - 9]$

$11.p \rightarrow q \vee \quad 1, 2 - 6, 7 - 10$

6.4 Negation Introduction and Proof by Contradiction

6.4.1 Negation Introduction

$$\dfrac{\begin{array}{|c|}\hline \phi \\ \vdots \\ \bot \\ \hline \end{array}}{\neg\phi}\neg i$$

6.4.2 Proof By Contradiction (Reductio Ad Absurdum)

$$\dfrac{\begin{array}{|c|}\hline \neg\phi \\ \vdots \\ \bot \\ \hline \end{array}}{\phi}PBC$$

Example: Prove the following sequent is valid.

$$p \rightarrow q, p \rightarrow \neg q \bot \neg p$$

Solution:

$1.p \to q$	premise
$2.p \to \neg q$	premise
$3.p$	assumption(3)
$4.q$	$\to e$ 3,1
$5.\neg q$	$\to e$ 3,2
$6.\bot$	$\neg e$ 4,5
$7.\neg p$	$\neg i$ 3-6

Example: Prove $p \to (q \to r), p, \neg r \vdash \neg q$ is valid, without using MT.
Solution:

$1.p \to (q \to r)$	premise
$2.p$	premise
$3.\neg r$	premise
$4.q$	assumption
$5.q \to r$	$\to e$ 2,1
$6.r$	$\to e$ 4,5
$7.\bot$	$\neg e$ 3,6
$8.\neg q$	$\neg_i$ 4-7

7 Derived Rules

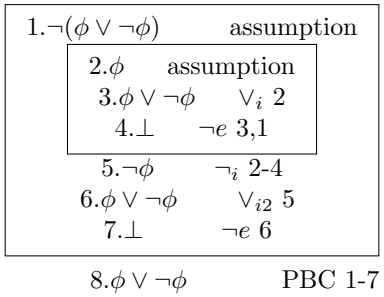
MT can be derived from $\to e, \neg e$ and $\neg_i$.

$1.\phi \to \psi$	premise
$2.\neg\psi$	premise
$3.\neg\phi$	assumption)
$4.\neg\psi$	$\to e$ 3,1
$5.\bot$	$\neg e$ 2,4
$6.\neg\phi$	$\neg_i 3-5$

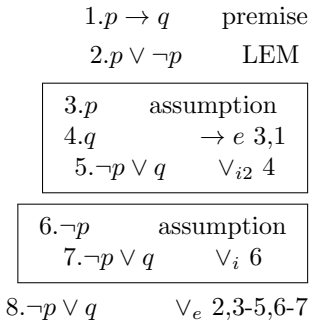
8 Law of the Excluded Middle (LEM)

- It is also called as Tertium non datur. (There is no third possibility)
- Says that $\phi \vee \neg\phi$ is always true.

$$\vdash \phi \vee \neg\phi$$



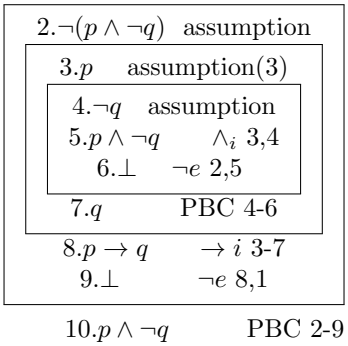
Example: Using LEM, show that $p \to q \vdash \neg p \vee q$ as valid.



9 Provable Equivalence

- ϕ and ψ are provable equivalent if and only if the sequents $\phi \vdash \psi$ and $\psi \vdash \phi$ are valid.
- It is denoted by $\phi \dashv\vdash \psi$.
- Ultimately we could define the $\phi \dashv\vdash \psi$ as $\vdash (\phi \to \psi) \wedge (\psi \to \phi)$.

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



BITS Pilani Hyderabad Campus
CS F214 Logic in Computer Science,
I Semester 2021-2022
Lecture Notes
Lecture 11-16

10 Well Formed Formulas

10.1 What is allowable formula ϕ?

- Formulas are string over the alphabet $\{p, q, r..\} \cup \{p_1, p_2, ...\} \cup \{\neg, \wedge, \vee, \rightarrow, (,)\}$.
- But not all strings are admissible.
  e.g $(\neg)() \wedge pqr \rightarrow$.

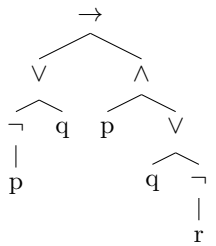
Definition : The well formed formulas of propositional logic are those which we obtain by using the construction rules below and only those by applying them finitely many times.

1. atom: Every propositional atom $p, q, r, ..$ or $p_1, p_2, p_3, ..$ is a well formed formula.
2. $\neg$: If ϕ is a well formed formula then so is $\neg\phi$.
3. $\wedge$: If ϕ and ψ are well formed formulas so is $\phi \wedge \psi$.
4. $\vee$: If ϕ and ψ are well formed formulas so is $\phi \vee \psi$.
5. $\rightarrow$: If ϕ and ψ are well formed formulas so is $\phi \rightarrow \psi$.

10.2 How do we show that a formula is well formed?

Construct in a top down manner, a parse tree where its leaves are atoms. e.g.

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



- Parse trees of well formed formulas are either an atom as root or the root contains $\neg, \wedge, \vee$ or $\rightarrow$.

- In case of $\neg$, there is only one sub-tree coming out of the root. In case of $\vee, \wedge, \rightarrow$ these are forms.
- A Sub-formula of a formula corresponds to a sub-tree of the parse tree.
  - You can obtain the formula back by using In-order traversal of the parse tree.
  - In-order traversal can be obtained by recursively printing left sub-tree, printing root, printing right sub-tree.
  - Pre-order traversal can be obtained by recursively printing root, printing left sub-tree, printing right sub-tree.
  - Post-order traversal can be obtained by recursively printing left sub-tree, printing right sub-tree, printing root.

10.3 Height of a Parse Tree:

Definition: Given a well-formed formula /phi, its height is defined to be 1 plus length of the longest path in the parse tree starting from the root.

Simple example of different arithmetic notations is as follows:

- x+y: Infix Notation
- xy+: Postfix Notation
- +xy: Prefix Notation

11 Mathematical Induction

- C.F. Gauss derived the formula of $\sum_{n=1}^{n} = \frac{n(n+1)}{2}$ as a child prodigy.
- One of the ways to prove this is by the use of mathematical induction.

Suppose, we wish to show $M(n), M \in N$.

1. **Base Case**: Natural number 1 has the property i.e. we have a proof of M(1).
2. **Inductive Hypothesis**: We assume that M(n) is true.
3. **Induction Step**: Prove that if $M(n)$ is true then $M(n+1)$ is true.

Theorem: *The sum of the first n natural numbers in equal to* $\frac{n(n+1)}{2}$

Proof: Let $M(n)$: the sum of the first n natural numbers not equal to $\frac{n(n+1)}{2}$.

- Base Case: To show that $M(1)$ is true.

  The sum of first one natural number is 1.
  Furthermore $M(1)$ states that the sum of first natural number equals $\frac{1(1+1)}{2} = 1$.

  $\therefore M(1)$ is true.

- Inductive Step: Suppose $M(n)$ is true. Consider $M(n+1)$: The sum of the first $(n+1)$ natural numbers is $\frac{(n+1)(n+2)}{2}$.

$$\sum_{i=1}^{n+1} i = \sum_{i=1}^{n} i + (n+1)$$

By the induction hypothesis, $M(n)$ is true.

$$\sum_{i=1}^{n+1} = \frac{n(n+1)}{2} + \frac{2(n+1)}{2}$$

$$\sum_{i=1}^{n+1} = \frac{(n+1)(n+2)}{2}$$

$\therefore M(n+1)$ is true.

12 Course of Values Induction or Strong Induction

1. **Base Case**: Natural number 1 has the property i.e. we have a proof of $M(1)$.
2. **Inductive Step**: We assume that $M(1) \wedge M(2) \wedge M(3).. \wedge M(n)$ is true and show that $M(n+1)$ is true.

Statements on parse trees are often shown by strong induction on height.
It is also called as structural Induction.

Theorem: *For every well-formed proposition logic formula, the number of left brackets equal the number of right brackets.*

Proof: Course of values induction on the height og the parse tree corresponding to the well-formed formula.

Let $M(n)$: All formulas of height n, have the same number of left and right brackets.

- **Base Case**: We will show that $M(1)$ is true.
  A parse tree of height 1 has only an atom and no bracket. Hence $M(1)$ is true.

- **Inductive Step**: Suppose $M(1), M(2), ...M(n)$ is true, we will show that $M(n+1)$ is true.
  A parse tree of height ≥ 2 has as its root either of $\neg, \vee, \wedge, \rightarrow$. Suppose the root as $\neg$. Then the sub-tree rooted at $\neg$ is of height n.
  By the induction hypothesis property as true for the formula ϕ corresponding to that of sub-tree. The formula corresponding to the full tree is $(\neg\phi)$, which has equal number of left and right brackets. Since we added one left bracket and one right bracket.

  Case (2): Let the root be $\vee$. If this i so, there exist two well formed formulas ϕ_1 and ϕ_2 so that the present formula is $(\phi_1 \vee \phi_2)$. The parse trees corresponding to the formulas ϕ_1 and ϕ_2 have height less than or equal to n.
  $\therefore$ by the induction hypothesis both these parse trees have equal number of left and right brackets each.
  $\therefore (\phi_1 \vee \phi_2)$ has equal number of left and right brackets. Since this adds one more left (right) bracket to the sum of left brackets of $(\phi_1 \vee \phi_2)$.

  Case (3), (4) correspond to the binary connectives $\wedge, \rightarrow$, for which the argument is similar.

13 Semantics of Propositional Logic

Def 1: *The set of truth values contains two elements - T and F, where T represents 'true' and F corresponds to 'false'.*

Def 2: *The valuation on model of formula ϕ is an assignment of each propositional atom in ϕ to a truth value. A truth table lists all valuations of a formula ϕ.*

Do all valid sequents preserve truth computed by our truth table semantics?

Def: *If for all the valuations in which all of $\phi_1, \phi_2, ..., \phi_n$ evaluate to T and formula χ also evaluates to T, we say that (the semantic entailment relation) $\phi_1, \phi_2, ..., \phi_n \vDash \chi$ holds and we call $\vDash$ as the semantic entailment relation.*

Examples:

1. $p \wedge q \vDash p$ holds.

p	q	$p \wedge q$	p	$p \vee q$
F	F	F	F	F
F	T	F	F	T
T	F	F	T	T
T	T	T	T	T

2. $p \vee q \vDash p$ does not holds.

3. $p \vDash q \vee \neg q$ holds.

13.1 Soundness of Propositional Logic

Theorem: *Let $\phi_1, \phi_2, ..., \phi_n$ and ψ be propositional logic formulas. If $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is valid, then $\phi_1, \phi_2, ..., \phi_n \vDash \psi$ holds.*

Proof: Since $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is valid, we know that there is a proof of ψ from the premises $\phi_1, \phi_2, ..., \phi_n$.

We will do course of values induction on the length of this proof (the numbering lines in it).

$M(R)$: For all the sequents $\phi_1, \phi_2, ..., \phi_n \vdash \psi, n \geq 0$, which have a proof of length R. It is the case that $\phi_1, \phi_2, ..., \phi_n \vDash \psi$ holds.

We will show $M(k)$ is true for $k \in N$ by course of values induction.
Example: Consider $p \wedge q \to r \vdash p \to q \to r$.

Solution:

$1.p \wedge q \to r$ premise

$2.p$	assumption
$3.q$	assumption
$4.p \wedge q$	$\wedge$,2,3
$5.r$	$\to$e 4,1
$6.q \to r$	$\to$i 3-5

$7.p \to (q \to r)$ $\to$i 2-6

1. **Base Case**: We wish to show M(1) is true.This is an example of sequent with one line proof.

$$1.\phi \quad \text{premise}$$

Above is a proof that shows $\phi \vdash \phi$.
$\phi \vDash \phi$ hols because whenever ϕ is true, ϕ is true. $\vdash \phi \vee \neg\phi$

$$1.\phi \vee \neg\phi \quad \text{LEM}$$

$\vDash \phi \vee \neg\phi$ For every valuation for ϕ is true, $\neg\phi$ is false. Therefore, $\phi \vee \neg\phi$ is true. Similarly, for every valuation for ϕ is false, $\neg\phi$ is true.
$\therefore$ for every valuation $\phi \vee \neg\phi$ is true.
$\therefore \vDash \phi \vee \neg\phi$ holds.

2. **Inductive Step**: We assume that shortest proof of the sequent $\phi_1, \phi_2, ..., \phi_n \vdash \psi$ is of length K. We assert the inductive hypothesis of all sequents that have a proof of length $<$ K.

A proof has the structure

$$\begin{aligned} 1.\ \phi_1 &\quad \text{premise} \\ &\quad ..... \\ n.\ \phi_n &\quad \text{premise} \\ K.\ \psi &\quad \text{justification} \end{aligned}$$

Two issues the proof needs to deal with.

(a) What happens in between. (We hope this will be solved by induction hypothesis).

(b) What is the last rule?(Proof needs to consider all such cases).

Cases (corresponding to which was the last rule applied)

(a) Consider $\wedge i$ to be the last rule applied.
ψ has to be of the form $\psi_1 \wedge \psi_2$ citing lines K_1 and K_2 respectively.
K_1 and K_2 ¡ K.
Lines 1-K_1 constitute a proof of the sequent. $\phi_1, \phi_2, ..., \phi_n \vdash \psi_1$.
Likewise, Lines 1-K_2 constitute a proof of the sequent. $\phi_1, \phi_2, ..., \phi_n \vdash \psi_2$.

Induction hypothesis is that $M(1), .., M(k-1)$ is true. By the induction hypothesis $\phi_1, \phi_2, ..., \phi_n \vDash \psi_1$ holds and $\phi_1, \phi_2, ..., \phi_n \vDash \psi_2$ holds.

$$1.\phi_1, \phi_2, ..., \phi_n \vDash \psi_1 2.\phi_1, \phi_2, ..., \phi_n \vDash \psi_2$$

BITS Pilani Hyderabad Campus
CS F214 Logic in Computer Science,
I Semester 2021-2022
Lecture Notes
Lecture 17-22

We wish to show $\phi_1, \phi_2, ..\phi_n \vDash \psi$ holds i.e. $\phi_1, \phi_2, ..\phi_n \vDash \psi_1 \wedge \psi_2$ holds.

From (1) and (2), we have that for each valuation for which $\phi_1, \phi_2, ..., \phi_n$ is true ψ_1 is true also ψ_2 is true.

$\therefore$ for each valuation $\psi_1 \wedge \psi_2$ is true.
$\therefore$ we have,

$$\phi_1, \phi_2, ..\phi_n \vDash \psi$$

Case 2: When last rule applied is $\vee_e$. We must have proved on have as a premise $\eta_1 \vee \eta_2$.
This means that the sequent $\phi_1, \phi_2, ..\phi_n \vdash \eta_1 \vee \eta_2$ is valid.

The first 'box' $\boxed{\begin{array}{c}\eta_1 \\ . \\ . \\ . \\ \psi\end{array}}$ gives us a proof of the sequent

$$\phi_1, \phi_2, ..\phi_n, \eta_1 \vdash \psi \qquad 4$$

likewise, the second 'boxes' gives us

$$\phi_1, \phi_2, ..\phi_n, \eta_2 \vdash \psi \qquad 5$$

By induction hypothesis, the corresponding semantic entailment relations for (3),(4) and (5) holds. We shall call semantic entailment relations as (6),(7) and (8).
Consider an arbitrary evaluation for which $\phi_1, ...\phi_n$ are true.
By (6), we have that $\eta_1 \vee \eta_2$ is true for this valuation. This means that for this valuation at least η_1, η_2 is true.

Case(2a) η_1 is true.
By (7) we have that ψ is true.

Case (2b) η_2 is true.
By (8), we have that ψ is true.
Therefore in both the cases ψ is true.
$\therefore wehave \phi_1, \phi_2, ...\phi_n \vDash \psi$ holds.

Rest of the cases can be shown similarly.

13.2 Contrapositive of The Soundness Theorem

If $\phi_1, ..\phi_n \vDash \psi$ does not hold then,$\phi_1, ..\phi_n \vdash \psi$ is not valid.

This implies that in order to show that a sequent is not valid, it suffices to find a valuation for which $\phi_1, ..\phi_n$ are true but ψ is false.

14 Completeness of Propositional Logic

Whenever $\phi_1, .., \phi_n \vDash \psi(1)$ holds then there exists a natural deduction proof for the sequent $\phi_1, .., \phi_n \vdash \psi$.

Proof Sketch

1. Assuming (1), we show that $\vDash \phi_1 \to (\phi_2 \to (..(\phi_n \to \psi))$ holds.
2. We show that $\vdash \phi_1 \to (\phi_2 \to (..(\phi_n \to \psi))$ is valid.
3. $\phi_1, .., \phi_n \vdash \psi$ is valid.

Please refer the textbook for the proof.

14.1 Corollary[Soundness and Completeness of Propositional Logic]

Let $\phi_1, ..., \phi_n$ and ψ be formulae of propositional logic. Then, $\phi_1, ..\phi_n \vdash \psi$ is valid.

14.2 Semantic Equivalence

Let ϕ and ψ be formulas in propositional logic. We say that, ϕ an ψ are semantically equivalent iff $\psi \vDash \psi$ holds and $\psi \vDash \phi$ holds as well.
We write $\phi \equiv \psi$
We call ϕ valid iff $\vDash$ holds. Semantic equivalence is identical to provable equivalence.
e.g. $p \to \equiv \neg q \to \neg p$
$p \to q \equiv \neg p \vee q$

We want to transform formulaes into forms in which validity checks are easy.
$\phi \to \psi \equiv \neg\phi \vee \psi$.

Definition: *A literal L is either an atom or negation of atom.*

A formula C in Conjunctive Normal Form (CNF) if it is a conjunction of the

clauses where each clause is a disjunction of literals.
e.g.

$$(1)\ (p \vee q) \wedge (\neg p \vee r)(2)(\neg(q \vee \neg p) \vee r) \wedge (p \vee q)$$

(2) is not in CNF form as it has negation of clause.

Definition of CNF in Backus Norm Form (BNF)

$$\begin{aligned} \text{Literal} \quad & L ::= p|\neg p \\ \text{Clause} \quad & D ::\neq L|L \vee D \\ \text{CNF Formula} \quad & C ::= D|D \wedge C \end{aligned}$$

Observations:

1. A CNF is a conjunction of clauses $C_1, C_2, ..., C_n$
   i.e. $C \equiv C_1 \wedge C_2 \wedge ... \wedge C_n$.
   For C to be true it must be the case that each one of $C_1, C_2, ..., C_n$ are true.
   Suppose C_i is not a valid formula then C is not valid. Now there may be a single clause featuring all n atoms.

Lemma:*A disjunction of literals $L_1 \vee L_2 \vee .... \vee L_m$ is valid iff there are i,j such that $1 \leq i, j \leq m$, so that L_i is $\neg L_j$*
Proof: Consider i,j so that L_i is $\neg L_j$.

Now,

$$L_1 \vee L_2 \vee .. \vee L_m \equiv (L_1 \vee ...) \vee (L_i \vee \neg L_i) \quad (1)$$

Now, $(L_i \vee \neg L_i)$ is always true.
Suppose L_i is an atom P. In any valuations, p is either true or false.

Case 1: p is true
Then L_i is true.
$\therefore (L_i \vee \neg L_i)$ is true.
$\therefore$ (1) is true.

Case 2: p is false.
L_i is false, $\neg L_i$ is true.
$\therefore (L_i \vee \neg L_i)$ is true.
$\therefore$ (1) is true.
Suppose L_i is $\neg p$, the formula is valid for similar reasons.

To prove the converse, suppose for all i,j $1 \leq i, j \leq m$. L_i is not $\neg L_j$ then
The formula is $L_1 \vee L_2 \vee ... \vee L_m$.
Now, consider a valuation where each literal is made false. Suppose a literal L_i is P_k. Then set P_k false in the valuation.
Suppose L_i is $\neg P_l$. Then set P_l to true in the valuation.
This procedure will not make any literal evaluate to false because that would imply that such a literal is a negation of a literal that had previously been made false.

Definition: *Given a formula ϕ in propositional logic, we say that ϕ is satisfiable if it has a validation in which it evaluates to true.*

e.g.

$$(1) p \vee q \rightarrow p \text{ is satisfiable}$$
$$(2) (p \vee q) \wedge (\neg p) \wedge (\neg q)$$

Proposition: Let ϕ be a formula of propositional logic. Then ϕ is satisfiable iff $\neg\phi$ is not valid.

Proof: Suppose ϕ is satisfiable.
Then there exist a valuation of ϕ, in which ϕ evaluates to true. In this valuation $\neg\phi$ evaluates to false.
$\therefore \neg\phi$ is not valid.

To prove the converse, suppose $\neg\phi$ is not valid. Then, there exists a valuation for which $\neg\phi$ is false.
For the same valuation, we have ϕ evaluates to True.
Since $\phi \equiv \neg\neg\phi$.

$\therefore \phi$ is satisfiable.

Please read sec 1.5.2 from textbook regarding the conversion of any formula in CNF (Page 57).

The logical constants ('bottom') $\perp$ and ('top') $\top$ denote respectively unsatisfiable formula and tautology.

15 Horn Formula

A Horn formula is a formula ϕ in propositional logic, if it can be generated as instance of the following grammar.

$$P ::= \bot|\top|P$$
$$A ::= P|P \wedge A$$
$$\text{Horn Clause, } C ::= A \rightarrow P$$
$$\text{Horn Formula, } H ::= C|C \wedge H$$

Formula	Explanation
1.$(p \wedge q \wedge r \rightarrow p) \wedge (q \wedge s \rightarrow p) \wedge (\top \rightarrow s) \wedge (r \wedge s \rightarrow \bot) \wedge (\bot \wedge p \rightarrow r)$	Horn Formula
2.$(p \wedge q \wedge r \rightarrow \neg p) \wedge (q \wedge r \rightarrow q)$	Not Horn Formula due to $\neg p$
3. $(p \wedge r \wedge r \rightarrow \bot) \wedge (\neg q \wedge r \rightarrow p)$	Not Horn Formula due to $\neg q$
4. $(p_1 \wedge p_2 \wedge p_3 \rightarrow (p_4 \wedge p_5)) \wedge (\top \rightarrow p_5)$	Not Horn Formula due to $p_4 \wedge p_5$
5. $(p \wedge q \rightarrow r) \wedge (p \wedge q) \wedge (r \vee s \rightarrow p)$	Not Horn Formula due to $\vee$ and $p \wedge q$

15.1 Deciding Satisfiability of Horn Formula

- Maintain a list of all occurrences of type P in your formula.
  1. It marks $\top$, if it occurs in that list.
  2. If there is a conjunct $p_1 \wedge p_2 \wedge .. \wedge p_k \rightarrow p'$ of ϕ such that all p_j with $1 \leq j < k$ is marked, then mark p' as well and go to step 2. Otherwise if there is no such conjunct go to step 3.
  3. If $\bot$ is marked, print 'Unsatisfiable' and Stop. Otherwise go to step 4.
  4. Print 'Satisfiable'.

Example

1.$(p \wedge q \wedge w \rightarrow \bot) \wedge (t \rightarrow \bot) \wedge (r \rightarrow p) \wedge (\top \rightarrow r) \wedge (\top \rightarrow q) \wedge (u \rightarrow s)$

Solution:

- Mark all occurrences of $\top$.
- Mark r, q, u.
- Mark p.

- Mark S.
- Print Satisfiable.

Example

2.$(p_5 \rightarrow p_{11}) \wedge (p_2 \wedge p_3 \wedge p_5 \rightarrow p_{13}) \wedge (\top \rightarrow p_5) \wedge (p_5 \wedge p_{11} \rightarrow \bot)$

Solution:

- Mark all occurrences of $\top$.
- Mark $p_5, p_1 1, \bot$
- Print 'Unsatisfiable'.

Example: **1(a).Construct a proposition for the given truth table.**

p	q	ϕ
F	F	T
F	T	T
T	F	F
T	T	F

- The disjunctive normal form formula can be constructed as $(\neg p \wedge \neg q) \vee (\neg p \wedge q)$
- In this example, we have considered all the cases for which the proposition is true and created set of clauses by anding them.
- Later all the clauses are combined using disjunction.

Example: **1(b).Prove the converse of the previous example.**

- The DNF clause consists of set of clauses combined using disjunction.
- All these clauses have set of atoms which are combined using conjunction.
- If the truth table is true then there is at least one clause with all atoms true.
- Now, for all the cases where the proposition evaluates to false there is at least one atom which differs to the evaluation where the formula is true.
- $\therefore$ the corresponding DNF clauses also evaluates to false as at least one of the atom in the clause evaluates to false.
- Thus, DNF becomes disjunction of all clauses which evaluates to false and thus formula evaluates to false.

Example: **2.Convert given truth table to CNF formula.**

Solution:

- Consider truth table for $\neg\phi$.
- Construct proposition in DNF for $\neg\phi$
- Take negation of DNF formula and apply Demorgan's Law to convert it into CNF.

p	q	$\neg\phi$
F	F	F
F	T	F
T	F	T
T	T	T

$$\neg\phi \equiv (p \wedge \neg q) \vee (p \wedge q)$$
$$\phi \equiv \neg(p \wedge \neg q) \wedge \neg(p \wedge q)$$
$$\boxed{\phi \equiv (\neg p \vee q) \wedge (\neg p \vee \neg q)}$$

CHAPTER 2 PREDICATE LOGIC

Express things like
- "for all"
- "there exists"

Satisfiability: Does there exist a valuation for which the formula evaluates to true

Validity: Is the formula true for all valuations?

PREDICATES OR PROPOSITIONAL FUNCTIONS

$S(x)$: x is a student

$I(x)$: x is an instructor

$Y(x, y)$: x is younger than y.

S(Rohit) = True

S(Venkat) = False

Unary predicate

binary predicate

Every student x is younger than some instructor y.'

Quantifiers : ① Universal Quantifier "For all" $\forall$

② Existential quantifier : "There exists"

"Everybody is a student" $\forall x\, S(x)$

Every student x is younger than some instructor y.'

(1)

$$\forall x\,(S(x) \rightarrow (\exists y\,(I(y) \wedge Y(x,y)))).$$

For every x, if x is a student, then there is some y which is an instructor such that x is younger than y.

(2) $\forall x \left[S(x) \rightarrow \left(\exists y \left(I(y) \rightarrow Y(x,y) \right) \right) \right]$

Not all birds can fly.

For that we choose the predicates B and F which have one argument expressing

$B(x)$: x is a bird

$F(x)$: x can fly.

The sentence 'Not all birds can fly' can now be coded as

$$\neg(\forall x\,(B(x) \rightarrow F(x)))$$

saying: 'It is not the case that all things which are birds can fly.' Alternatively, we could code this as

$$\exists x\,(B(x) \wedge \neg F(x))$$

younger than

Every child is younger than its mother.

Using predicates, we could express this sentence as

$$\forall x \forall y \, (C(x) \wedge M(y, x) \rightarrow Y(x, y))$$

$C(x)$: x is a child

$M(y,x)$: y is x's mother

$Y(x,y)$: x is younger than y

→ However, every child has a unique mother.

$m(x)$: "returns" mother of x

$\forall x \, [C(x) \rightarrow Y(x, m(x))]$

Define 2 kinds of things

— "Objects" — variables, constants, functions such as $m(x)$ which we will call terms

— Predicates.

Definition 2.1 Terms are defined as follows.

- Any variable is a term.
- If $c \in \mathcal{F}$ is a nullary function, then c is a term.
- If $t_1, t_2, \ldots, t_n$ are terms and $f \in \mathcal{F}$ has arity $n > 0$, then $f(t_1, t_2, \ldots, t_n)$ is a term.
- Nothing else is a term.

In Backus Naur form we may write

$$t ::= x \mid c \mid f(t, \ldots, t)$$

n-ary

where x ranges over a set of variables var, c over nullary function symbols in $\mathcal{F}$, and f over those elements of $\mathcal{F}$ with arity $n > 0$.

S (Rohit) = true

Definition 2.3 We define the set of formulas over $(\mathcal{F}, \mathcal{P})$ inductively, using the already defined set of terms over $\mathcal{F}$:

- If $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 1$, and if $t_1, t_2, \ldots, t_n$ are terms over $\mathcal{F}$, then $P(t_1, t_2, \ldots, t_n)$ is a formula.
- If ϕ is a formula, then so is $(\neg\phi)$.
- If ϕ and ψ are formulas, then so are $(\phi \wedge \psi)$, $(\phi \vee \psi)$ and $(\phi \rightarrow \psi)$.
- If ϕ is a formula and x is a variable, then $(\forall x\, \phi)$ and $(\exists x\, \phi)$ are formulas.
- Nothing else is a formula.

Note how the arguments given to predicates are always terms. This can also be seen in the Backus Naur form (BNF) for predicate logic:

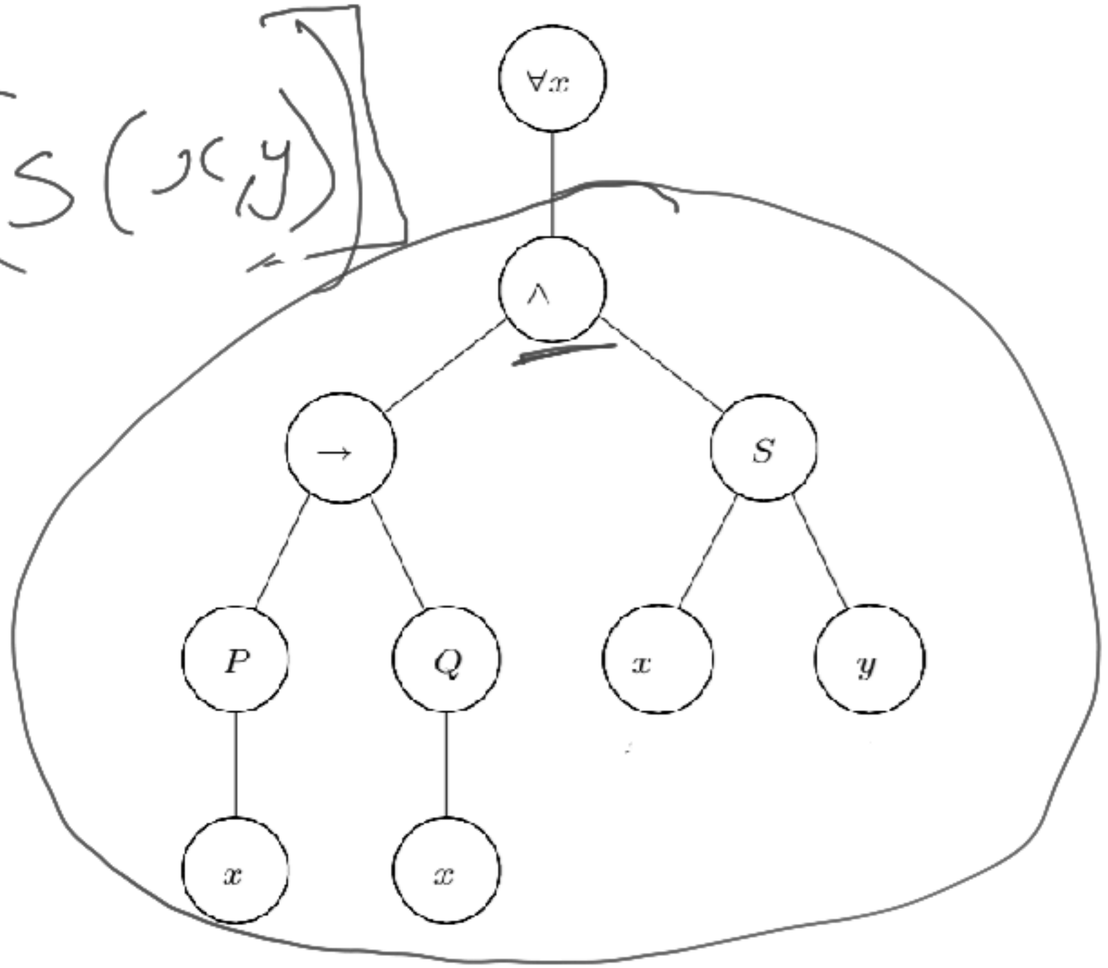
$$\phi ::= P(t_1, t_2, \ldots, t_n) \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid (\forall x\, \phi) \mid (\exists x\, \phi) \tag{2.2}$$

where $P \in \mathcal{P}$ is a predicate symbol of arity $n \geq 1$, t_i are terms over $\mathcal{F}$ and x is a variable. Recall that each occurrence of ϕ on the right-hand side of the $::=$ stands for any formula already constructed by these rules. (What role could predicate symbols of arity 0 play?)

Convention 2.4 For convenience, we retain the usual binding priorities agreed upon in Convention 1.3 and add that $\forall y$ and $\exists y$ bind like $\neg$. Thus, the order is:

- $\neg$, $\forall y$ and $\exists y$ bind most tightly;
- then $\vee$ and $\wedge$;
- then $\rightarrow$, which is right-associative.

$\forall x [(P(x) \rightarrow Q(x)) \wedge (S(x,y))]$



Example 2.5 Consider translating the sentence

Every son of my father is my brother.

into predicate logic. As before, the design choice is whether we represent 'father' as a predicate or as a function symbol.

1. As a predicate. We choose a constant m for 'me' or 'I,' so m is a term, and we choose further $\{S, F, B\}$ as the set of predicates with meanings

$$\begin{aligned} S(x,y): &\quad x \text{ is a son of } y \\ F(x,y): &\quad x \text{ is the father of } y \\ B(x,y): &\quad x \text{ is a brother of } y. \end{aligned}$$

Then the symbolic encoding of the sentence above is

$$\forall x \, \forall y \, (F(x,m) \wedge S(y,x) \rightarrow B(y,m)) \tag{2.3}$$

saying: 'For all x and all y, if x is a father of m and if y is a son of x, then y is a brother of m.'

2. As a function. We keep m, S and B as above and write f for the function which, given an argument, returns the corresponding father. Note that this works only because fathers are unique and always defined, so f really is a function as opposed to a mere relation.
The symbolic encoding of the sentence above is now

$$\forall x\,(S(x, f(m)) \rightarrow B(x, m)) \tag{2.4}$$

meaning: 'For all x, if x is a son of the father of m, then x is a brother of m;' it is less complex because it involves only one quantifier.

Domain-specific knowledge:

B(m,m)? Not well defined.

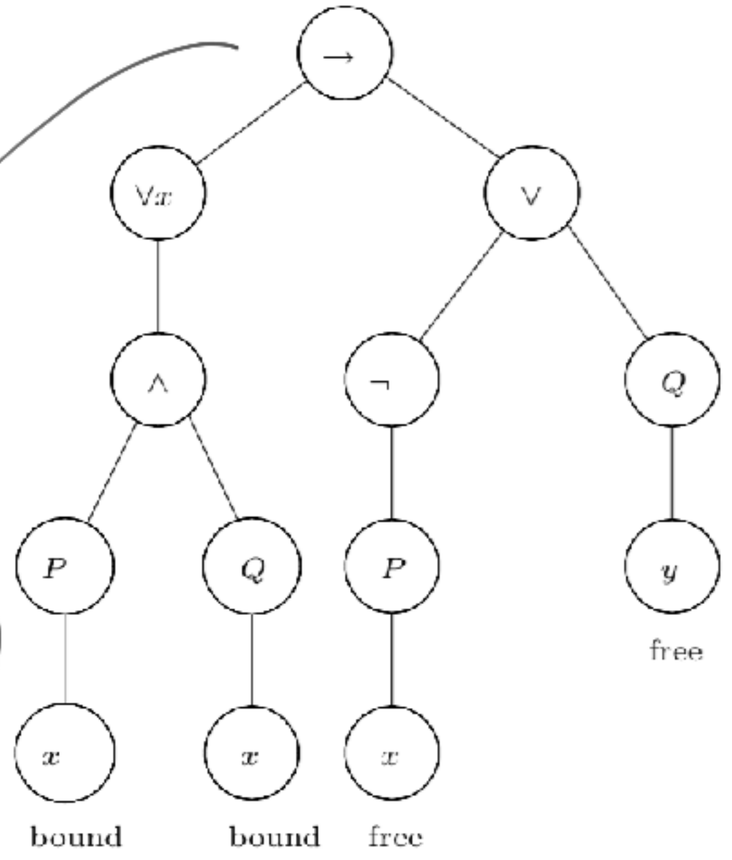
FREE & BOUND VARIABLES

$\forall x [P(x) \wedge (\exists x Q(x)$

Definition 2.6 Let ϕ be a formula in predicate logic. An occurrence of x in ϕ is free in ϕ if it is a leaf node in the parse tree of ϕ such that there is no path upwards from that node x to a node $\forall x$ or $\exists x$. Otherwise, that occurrence of x is called bound. For $\forall x\, \phi$, or $\exists x\, \phi$, we say that ϕ – minus any of ϕ's subformulas $\exists x\, \psi$, or $\forall x\, \psi$ – is the scope of $\forall x$, respectively $\exists x$.

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

$[\forall x (P(x) \wedge Q(x))] \rightarrow [\neg P(x) \vee Q(y)]$

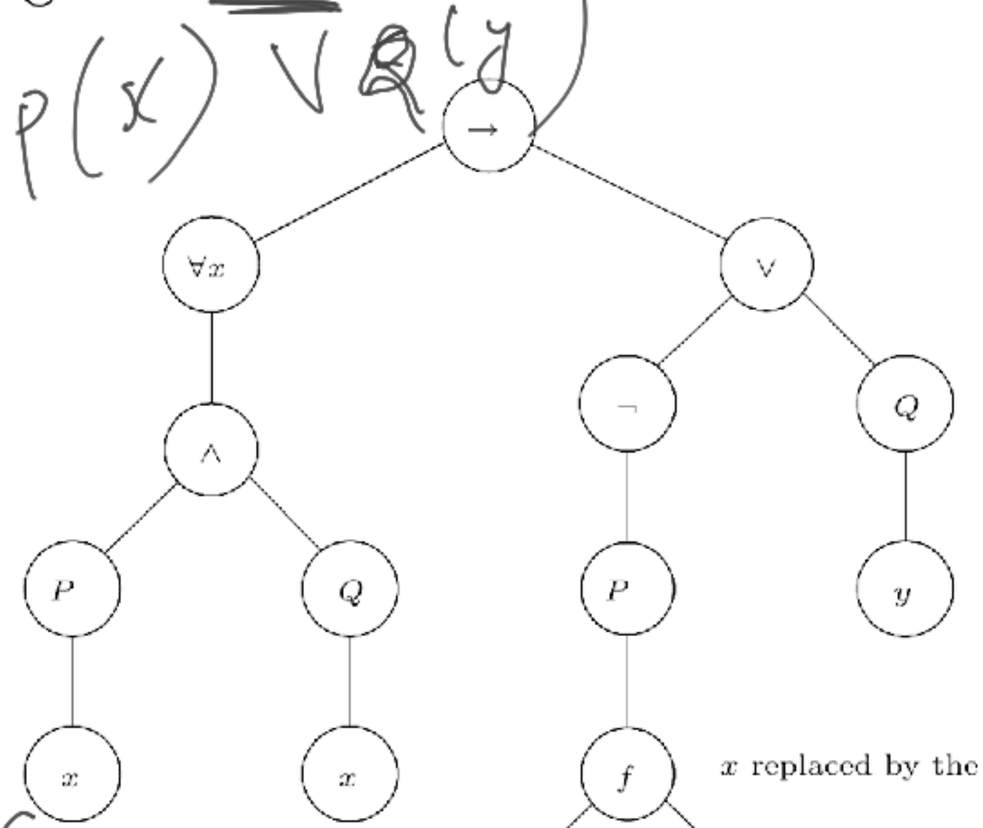


SUBSTITUTION

Definition 2.7 Given a variable x, a term t and a formula ϕ we define $\phi[t/x]$ to be the formula obtained by replacing each free occurrence of variable x in ϕ with t.

$\phi(x)$

$\equiv [\forall x(P(x) \wedge Q(x))] \rightarrow \neg P(x) \vee Q(y)$



x replaced by the term $f(x, y)$

$\phi(f(x,y)/x)$

$\equiv [\forall x(P(x) \wedge Q(x))] \rightarrow \neg P(f(x,y)) \vee Q(y)$

Definition 2.8 Given a term t, a variable x and a formula ϕ, we say that t is free for x in ϕ if no free x leaf in ϕ occurs in the scope of $\forall y$ or $\exists y$ for any variable y occurring in t.

ϕ :

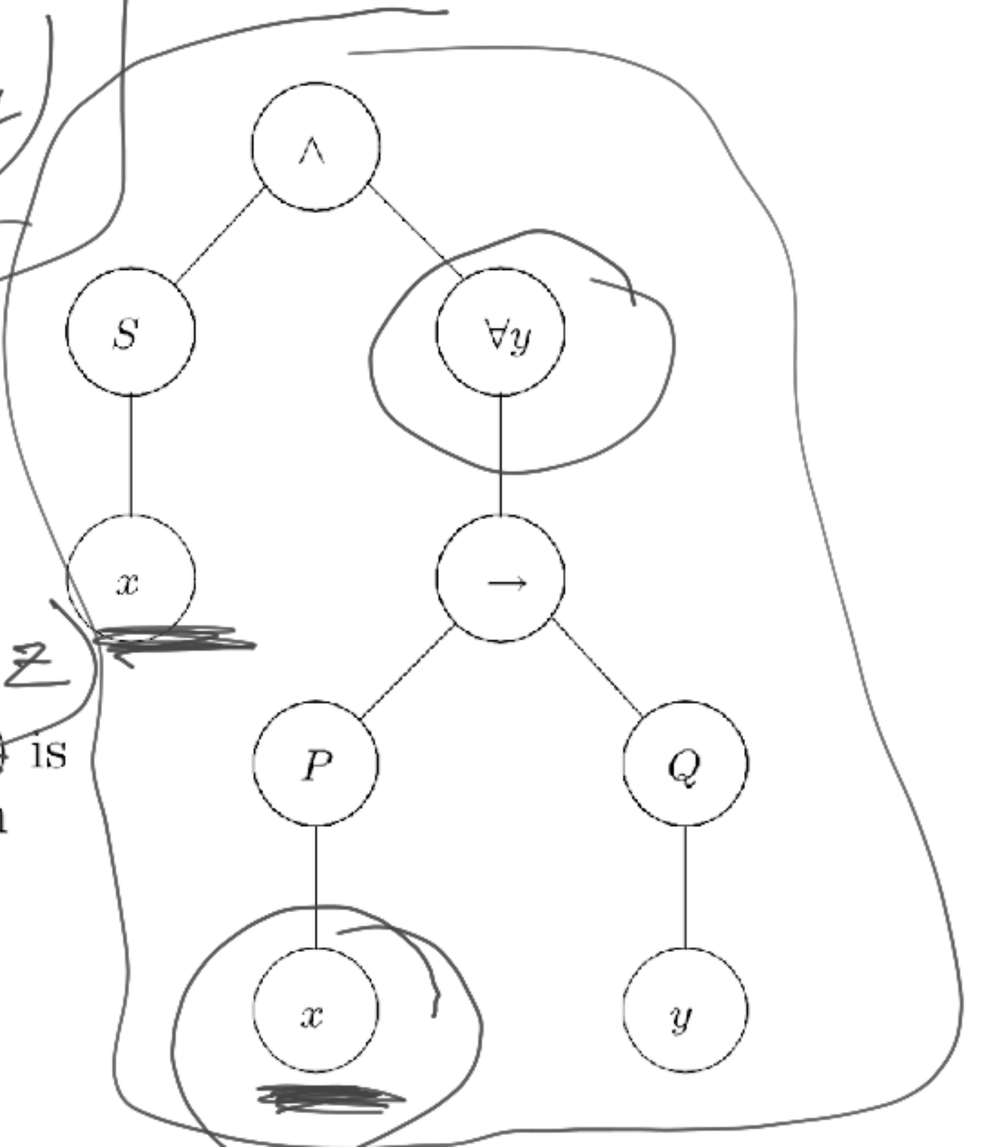
$S(x) \wedge \forall y [P(x) \rightarrow Q(y)]$

suppose $f(y,z)$ is a term

$\phi(f(y,y)/x)$

$f(z,z)$

the term ~~$f(y,y)$~~ is ~~not~~ free for x in this formula



Important to avoid undesirable capture of variables.

PROOF THEORY OF PREDICATE LOGIC

PROOF RULES FOR EQUALITY

NATURAL DEDUCTION RULES

EQUALS INTRODUCTION

$$\frac{}{t = t} =\text{i}$$

May only be invoked if t is a term !

EQUALS ELIMINATION

$$\frac{t_1 = t_2 \quad \phi[t_1/x]}{\phi[t_2/x]} =\text{e}.$$

Convention 2.10 Throughout this section, when we write a substitution in the form $\phi[t/x]$, we implicitly assume that t is free for x in ϕ; for, as we saw in the last section, a substitution doesn't make sense otherwise.

$\phi(y) : (y > 1) \to (y > 0)$

We obtain proof

1	$(x + 1) = (1 + x)$	premise	
2	$(x + 1 > 1) \to (x + 1 > 0)$	premise	
3	$(1 + x > 1) \to (1 + x > 0)$	=e 1, 2	

establishing the validity of the sequent

$$x + 1 = 1 + x,\ (x + 1 > 1) \to (x + 1 > 0) \ \vdash\ (1 + x) > 1 \to (1 + x) > 0.$$

$$t_1 = t_2 \vdash t_2 = t_1 \tag{2.6}$$

$$t_1 = t_2,\ t_2 = t_3 \vdash t_1 = t_3. \tag{2.7}$$

A proof for (2.6) is:

1	$t_1 = t_2$	premise
2	$t_1 = t_1$	=i
3	$t_2 = t_1$	=e 1, 2

where ϕ is $x = t_1$. A proof for (2.7) is:

1	$t_2 = t_3$	premise
2	$t_1 = t_2$	premise
3	$t_1 = t_3$	=e 1, 2

where ϕ is $t_1 = x$, so in line 2 we have $\phi[t_2/x]$ and in line 3 we obtain $\phi[t_3/x]$, as given by the rule =e applied to lines 1 and 2. Notice how we applied the scheme =e with several different instantiations.

Universal Quantifier Elimination

$$\frac{\forall x\, \phi}{\phi[t/x]}\ \forall x\,\mathrm{e}.$$

Universal Quantifier Introduction

x_0 is a "fresh" variable.

$$\dfrac{\boxed{\begin{array}{l} x_0 \\ \vdots \\ \phi[x_0/x] \end{array}}}{\forall x\, \phi}\ \forall x\,\mathrm{i}.$$

proof of the sequent $\forall x\,(P(x) \to Q(x)),\ \forall x\,P(x) \vdash \forall x\,Q(x)$:

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\forall x\,P(x)$	premise
3	x_0	$P(x_0) \to Q(x_0)$	$\forall x\,\mathrm{e}\ 1$
4		$P(x_0)$	$\forall x\,\mathrm{e}\ 2$
5		$Q(x_0)$	$\to\mathrm{e}\ 3, 4$
6		$\forall x\,Q(x)$	$\forall x\,\mathrm{i}\ 3{-}5$

$$P(t),\ \forall x\,(P(x) \to \neg Q(x)) \ \vdash\ \neg Q(t)$$

1	$P(t)$	premise
2	$\forall x\,(P(x) \to \neg Q(x))$	premise
3	$P(t) \to \neg Q(t)$	$\forall x$ e 2
4	$\neg Q(t)$	$\to$e **3**, 1

PROOF RULES FOR EXISTENTIAL QUANTIFICATION

EXISTENTIAL QUANTIFIER INTRODUCTION

$$\frac{\phi[t/x]}{\exists x \phi}\ \exists x\ \mathrm{i}.$$

EXISTENTIAL QUANTIFIER ELIMINATION

$$\dfrac{\exists x\,\phi \qquad \boxed{\begin{array}{c} x_0\ \phi[x_0/x] \\ \vdots \\ \chi \end{array}}}{\chi}\ \exists e.$$

has to be indep of x_0!

x cannot depend on x_0.

$\forall x\,\phi \vdash \exists x\,\phi.$

1		$\forall x\,\phi$	premise
2		$\phi[x/x]$	$\forall x$ e 1
3		$\exists x\,\phi$	$\exists x$ i 2

$\forall x\,(P(x) \to Q(x)),\quad \exists x\,P(x) \quad \vdash \quad \exists x\,Q(x)$

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\exists x\,P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \to Q(x_0)$	$\forall x$ e 1
5		$Q(x_0)$	$\to$e 4, 3
6		$\exists x\,Q(x)$	$\exists x$ i 5
7		$\exists x\,Q(x)$	$\exists x$ e 2, 3–6

WRONG !

1		$\forall x\,(P(x) \to Q(x))$	premise
2		$\exists x\,P(x)$	premise
3	x_0	$P(x_0)$	assumption
4		$P(x_0) \to Q(x_0)$	$\forall x$ e 1
5		$Q(x_0)$	$\to$e 4, 3
6		$Q(x_0)$	$\exists x$ e 2, 3–5
7		$\exists x\,Q(x)$	$\exists x$ i 6

not independent of x_0!

$$\forall x\,(Q(x) \to R(x)),\ \exists x\,(P(x) \wedge Q(x)) \ \vdash\ \exists x\,(P(x) \wedge R(x))$$

1		$\forall x\,(Q(x) \to R(x))$	premise
2		$\exists x\,(P(x) \wedge Q(x))$	premise
3	x_0	$P(x_0) \wedge Q(x_0)$	assumption
4		$Q(x_0) \to R(x_0)$	$\forall x$ e 1
5		$Q(x_0)$	$\wedge$e$_2$ 3
6		$R(x_0)$	$\to$e 4, 5
7		$P(x_0)$	$\wedge$e$_1$ 3
8		$P(x_0) \wedge R(x_0)$	$\wedge$i 7, 6
9		$\exists x\,(P(x) \wedge R(x))$	$\exists x$ i 8
10		$\exists x\,(P(x) \wedge R(x))$	$\exists x$ e 2, 3–9

$\exists x\, Q(x)$

QUANTIFIER EQUIVALENCES

$\neg\forall x\, P(x) \vdash \exists x\, \neg P(x)$

1		$\neg\forall x\, P(x)$	premise
2		$\neg\exists x\, \neg P(x)$	assumption
3	x_0		
4		$\neg P(x_0)$	assumption
5		$\exists x \neg P(x)$	$\exists x$ i 4
6		$\bot$	$\neg$e 5, 2
7		$P(x_0)$	PBC 4–6
8		$\forall x\, P(x)$	$\forall x$ i 3–7
9		$\bot$	$\neg$e 8, 1
10		$\exists x\, \neg P(x)$	PBC 2–9

$$\neg\forall x\,\phi \vdash \exists x\,\neg\phi$$

1		$\neg\forall x\,\phi$	premise
2		$\neg\exists x\,\neg\phi$	assumption
3		x_0	
4		$\neg\phi[x_0/x]$	assumption
5		$\exists x\neg\phi$	$\exists x$ i 4
6		$\bot$	$\neg$e 5, 2
7		$\phi[x_0/x]$	PBC 4–6
8		$\forall x\,\phi$	$\forall x$ i 3–7
9		$\bot$	$\neg$e 8, 1
10		$\exists x\,\neg\phi$	PBC 2–9

RAA assumption

RAA – Reductio ad absurdum

$$\exists x\, \neg\phi \vdash \neg\forall x\, \phi$$

premise

1		$\exists x\, \neg\phi$	~~assumption~~
2		$\forall x\, \phi$	assumption
3		x_0	
4		$\neg\phi[x_0/x]$	assumption
5		$\phi[x_0/x]$	$\forall x$ e 2
6		$\bot$	$\neg$e 5, 4
7		$\bot$	$\exists x$ e 1, 3–6
8		$\neg\forall x\, \phi$	$\neg$i 2–7

i.e. we have $\neg\forall x\,\phi \vdash \exists x\,\neg\phi$

$\therefore \forall x\, \phi \wedge \psi \vdash \forall x\, (\phi \wedge \psi)$

1		$(\forall x\, \phi) \wedge \psi$	premise
2		$\forall x\, \phi$	$\wedge \mathrm{e}_1$ 1
3		ψ	$\wedge \mathrm{e}_2$ 1
4	x_0		
5		$\phi[x_0/x]$	$\forall x$ e 2
6		$\phi[x_0/x] \wedge \psi$	$\wedge$i 5, 3
7		$(\phi \wedge \psi)[x_0/x]$	identical to 6, since x not free in ψ
8		$\forall x\, (\phi \wedge \psi)$	$\forall x$ i 4–7

1		$\forall x\,(\phi \wedge \psi)$	premise
2	x_0		
3		$(\phi \wedge \psi)[x_0/x]$	$\forall x$ e 1
4		$\phi[x_0/x] \wedge \psi$	identical to 3, since x not free in ψ
5		ψ	$\wedge \mathrm{e}_2$ 3
6		$\phi[x_0/x]$	$\wedge \mathrm{e}_1$ 3
7		$\forall x\,\phi$	$\forall x$ i 2–6
8		$(\forall x\,\phi) \wedge \psi$	$\wedge$i 7, 5

Not rigorously correct.

RIGOROUS WAY

11. $\forall x \phi \wedge \psi$ ∧i

7, 10

1		$\forall x\,(\phi \wedge \psi)$	premise
2		x_0	
3		$(\phi \wedge \psi)[x_0/x]$	$\forall x$ e 1
4		$\phi[x_0/x] \wedge \psi$	identical to 3, since x not free in ψ
5		ψ	$\wedge e_2$ 3
6		$\phi[x_0/x]$	$\wedge e_1$ 3
7		$\forall x\,\phi$	$\forall x$ i 2–6
~~8~~		~~$(\forall x\,\phi) \wedge \psi$~~	~~∧i 7, 5~~

8 $(\phi \wedge \psi)(x_1/x)$ $\forall x$ e 1

9 $\phi(x_1/x) \wedge \psi$

10 ψ $\wedge e_2$ 9

$$(\exists x\,\phi) \vee (\exists x\,\psi) \vdash \exists x\,(\phi \vee \psi)$$

1		$(\exists x\,\phi) \vee (\exists x\,\psi)$			premise
2		$\exists x\,\phi$		$\exists x\,\psi$	assumpt.
3	x_0	$\phi[x_0/x]$	x_0	$\psi[x_0/x]$	assumpt.
4		$\phi[x_0/x] \vee \psi[x_0/x]$		$\phi[x_0/x] \vee \psi[x_0/x]$	$\vee$i 3
5		$(\phi \vee \psi)[x_0/x]$		$(\phi \vee \psi)[x_0/x]$	identical
6		$\exists x\,(\phi \vee \psi)$		$\exists x\,(\phi \vee \psi)$	$\exists x$ i 5
7		$\exists x\,(\phi \vee \psi)$		$\exists x\,(\phi \vee \psi)$	$\exists x$ e 2, 3–6
8		$\exists x\,(\phi \vee \psi)$			$\vee$e 1, 2–7

1		$\exists x\,(\phi \vee \psi)$		premise
2	x_0	$(\phi \vee \psi)[x_0/x]$		assumption
3		$\phi[x_0/x] \vee \psi[x_0/x]$		identical
4		$\phi[x_0/x]$	$\psi[x_0/x]$	assumption
5		$\exists x\,\phi$	$\exists x\,\psi$	$\exists x$ i 4
6		$\exists x\,\phi \vee \exists x\,\psi$	$\exists x\,\phi \vee \exists x\,\psi$	$\vee$**i** 5
7		$\exists x\,\phi \vee \exists x\,\psi$		$\vee$**e 3**, 4**–6**
8		$\exists x\,\phi \vee \exists x\,\psi$		$\exists x$ e **1**, 2**–7**

$\therefore \exists x(\phi \vee \psi) \vdash \exists x\phi \vee \exists x\psi$

1		$\exists x\, \exists y\, \phi$	premise
2	x_0	$(\exists y\, \phi)[x_0/x]$	assumption
3		$\exists y\, (\phi[x_0/x])$	identical, since x, y different variables
4	y_0	$\phi[x_0/x][y_0/y]$	assumption
5		$\phi[y_0/y][x_0/x]$	identical, since x, y, x_0, y_0 different variables
6		$\exists x\, \phi[y_0/y]$	$\exists x$ i 5
7		$\exists y\, \exists x\, \phi$	$\exists y$ i 6
8		$\exists y\, \exists x\, \phi$	$\exists y$ e3, 4–7
9		$\exists y\, \exists x\, \phi$	$\exists x$ e1, 2–8

Theorem 2.13 Let ϕ and ψ be formulas of predicate logic. Then we have the following equivalences:

1. (a) $\neg\forall x\,\phi \dashv\vdash \exists x\,\neg\phi$
   (b) $\neg\exists x\,\phi \dashv\vdash \forall x\,\neg\phi$.
2. Assuming that x is not free in ψ:
   (a) $\forall x\,\phi \wedge \psi \dashv\vdash \forall x\,(\phi \wedge \psi)$[3]
   (b) $\forall x\,\phi \vee \psi \dashv\vdash \forall x\,(\phi \vee \psi)$
   (c) $\exists x\,\phi \wedge \psi \dashv\vdash \exists x\,(\phi \wedge \psi)$
   (d) $\exists x\,\phi \vee \psi \dashv\vdash \exists x\,(\phi \vee \psi)$
   (e) $\forall x\,(\psi \to \phi) \dashv\vdash \psi \to \forall x\,\phi$
   (f) $\exists x\,(\phi \to \psi) \dashv\vdash \forall x\,\phi \to \psi$
   (g) $\forall x\,(\phi \to \psi) \dashv\vdash \exists x\,\phi \to \psi$
   (h) $\exists x\,(\psi \to \phi) \dashv\vdash \psi \to \exists x\,\phi$.
3. (a) $\forall x\,\phi \wedge \forall x\,\psi \dashv\vdash \forall x\,(\phi \wedge \psi)$
   (b) $\exists x\,\phi \vee \exists x\,\psi \dashv\vdash \exists x\,(\phi \vee \psi)$.
4. (a) $\forall x\,\forall y\,\phi \dashv\vdash \forall y\,\forall x\,\phi$
   (b) $\exists x\,\exists y\,\phi \dashv\vdash \exists y\,\exists x\,\phi$.

[3] Remember that $\forall x\,\phi \wedge \psi$ is implicitly bracketed as $(\forall x\,\phi) \wedge \psi$, by virtue of the binding priorities.

SEMANTICS OF PREDICATE LOGIC

MODELS

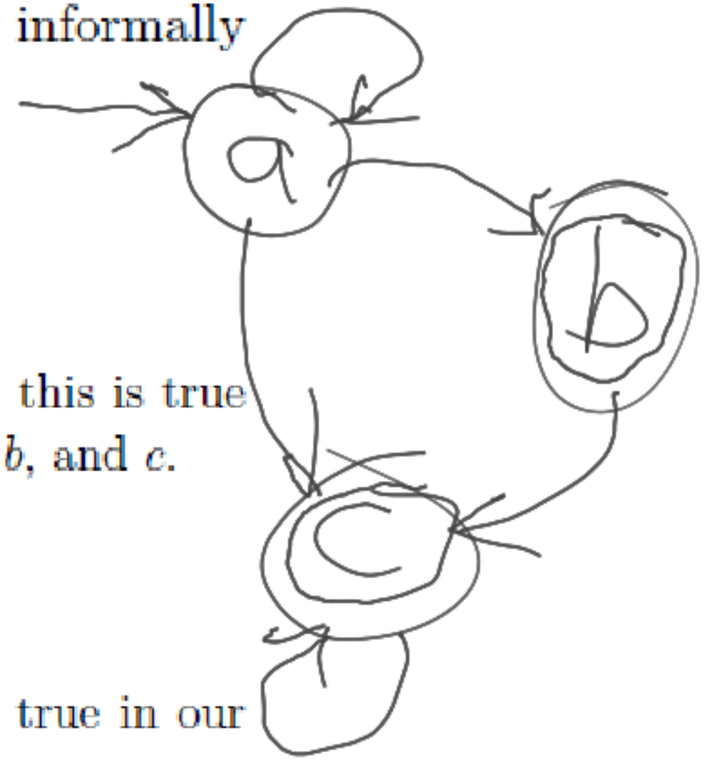
Definition 2.14 Let $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ a set of predicate symbols, each symbol with a fixed number of required arguments. A model $\mathcal{M}$ of the pair $(\mathcal{F}, \mathcal{P})$ consists of the following set of data:

1. A non-empty set A, the universe of concrete values;
2. for each nullary function symbol $f \in \mathcal{F}$, a concrete element $f^{\mathcal{M}}$ of A
3. for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^{\mathcal{M}} : A^n \to A$ from A^n, the set of n-tuples over A, to A; and
4. for each $P \in \mathcal{P}$ with arity $n > 0$, a subset $P^{\mathcal{M}} \subseteq A^n$ of n-tuples over A.

$P^{\mathcal{M}} : A^n \to \{T, F\}$

The distinction between f and $f^{\mathcal{M}}$ and between P and $P^{\mathcal{M}}$ is most important. The symbols f and P are just that: symbols, whereas $f^{\mathcal{M}}$ and $P^{\mathcal{M}}$ denote a concrete function (or element) and relation in a model $\mathcal{M}$, respectively.

Example 2.15 Let $\mathcal{F} \stackrel{\text{def}}{=} \{i\}$ and $\mathcal{P} \stackrel{\text{def}}{=} \{R, F\}$; where i is a constant, F a predicate symbol with one argument and R a predicate symbol with two arguments. A model $\mathcal{M}$ contains a set of concrete elements A – which may be a set of states of a computer program. The interpretations $i^{\mathcal{M}}$, $R^{\mathcal{M}}$, and $F^{\mathcal{M}}$ may then be a designated initial state, a state transition relation, and a set of final (accepting) states, respectively. For example, let $A \stackrel{\text{def}}{=} \{a, b, c\}$, $i^{\mathcal{M}} \stackrel{\text{def}}{=} a$, $R^{\mathcal{M}} \stackrel{\text{def}}{=} \{(a, a), (a, b), (a, c), (b, c), (c, c)\}$, and $F^{\mathcal{M}} \stackrel{\text{def}}{=} \{b, c\}$. We informally check some formulas of predicate logic for this model:



1. The formula

$$\exists y\, R(i, y)$$

says that there is a transition from the initial state to some state; this is true in our model, as there are transitions from the initial state a to a, b, and c.

2. The formula

$$\neg F(i)$$

states that the initial state is not a final, accepting state. This is true in our model as b and c are the only final states and a is the intitial one.

3. The formula

$$\forall x \forall y \forall z \, (R(x, y) \wedge R(x, z) \rightarrow y = z)$$

makes use of the equality predicate and states that the transition relation is deterministic: all transitions from any state can go to at most one state (there may be no transitions from a state as well). This is false in our model since state a has transitions to b and c.

4. The formula

$$\forall x \exists y \, R(x, y)$$

states that the model is free of states that deadlock: all states have a transition to some state. This is true in our model: a can move to a, b or c; and b and c can move to c.

Definition 2.17 A look-up table or environment for a universe A of concrete values is a function $l\colon \mathsf{var} \to A$ from the set of variables var to A. For such an l, we denote by $l[x \mapsto a]$ the look-up table which maps x to a and any other variable y to $l(y)$.

Definition 2.18 Given a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ and given an environment l, we define the satisfaction relation $\mathcal{M} \vDash_l \phi$ for each logical formula ϕ over the pair $(\mathcal{F}, \mathcal{P})$ and look-up table l by structural induction on ϕ. If $\mathcal{M} \vDash_l \phi$ holds, we say that ϕ computes to $\mathtt{T}$ in the model $\mathcal{M}$ with respect to the environment l.

P: If ϕ is of the form $P(t_1, t_2, \ldots, t_n)$, then we interpret the terms $t_1, t_2, \ldots, t_n$ in our set A by replacing all variables with their values according to l. In this way we compute concrete values $a_1, a_2, \ldots, a_n$ of A for each of these terms, where we interpret any function symbol $f \in \mathcal{F}$ by $f^{\mathcal{M}}$. Now $\mathcal{M} \vDash_l P(t_1, t_2, \ldots, t_n)$ holds iff $(a_1, a_2, \ldots, a_n)$ is in the set $P^{\mathcal{M}}$.

$\forall x$: The relation $\mathcal{M} \vDash_l \forall x\, \psi$ holds iff $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ holds for all $a \in A$.

$\exists x$: Dually, $\mathcal{M} \vDash_l \exists x\, \psi$ holds iff $\mathcal{M} \vDash_{l[x \mapsto a]} \psi$ holds for some $a \in A$.

$\neg$: The relation $\mathcal{M} \vDash_l \neg\psi$ holds iff it is not the case that $\mathcal{M} \vDash_l \psi$ holds.

$\vee$: The relation $\mathcal{M} \vDash_l \psi_1 \vee \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_1$ or $\mathcal{M} \vDash_l \psi_2$ holds.

$\wedge$: The relation $\mathcal{M} \vDash_l \psi_1 \wedge \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_1$ and $\mathcal{M} \vDash_l \psi_2$ hold.

$\rightarrow$: The relation $\mathcal{M} \vDash_l \psi_1 \rightarrow \psi_2$ holds iff $\mathcal{M} \vDash_l \psi_2$ holds whenever $\mathcal{M} \vDash_l \psi_1$ holds.

We sometimes write $\mathcal{M} \nvDash_l \phi$ to denote that $\mathcal{M} \vDash_l \phi$ does not hold.

Definition 2.20 Let Γ be a (possibly infinite) set of formulas in predicate logic and ψ a formula of predicate logic.

1. Semantic entailment $\Gamma \models \psi$ holds iff for all models $\mathcal{M}$ and look-up tables l, whenever $\mathcal{M} \models_l \phi$ holds for all $\phi \in \Gamma$, then $\mathcal{M} \models_l \psi$ holds as well.
2. Formula ψ is satisfiable iff there is some model $\mathcal{M}$ and some environment l such that $\mathcal{M} \models_l \psi$ holds.
3. Formula ψ is valid iff $\mathcal{M} \models_l \psi$ holds for all models $\mathcal{M}$ and environments l in which we can check ψ.
4. The set Γ is consistent or satisfiable iff there is a model $\mathcal{M}$ and a look-up table l such that $\mathcal{M} \models_l \phi$ holds for all $\phi \in \Gamma$.

Example 2.21 The justification of the semantic entailment

$$\forall x\,(P(x) \to Q(x)) \models \forall x\,P(x) \to \forall x\,Q(x)$$

is as follows. Let $\mathcal{M}$ be a model satisfying $\forall x\,(P(x) \to Q(x))$. We need to show that $\mathcal{M}$ satisfies $\forall x\,P(x) \to \forall x\,Q(x)$ as well. On inspecting the definition of $\mathcal{M} \models \psi_1 \to \psi_2$, we see that we are done if not every element of our model satisfies P. Otherwise, every element does satisfy P. But since $\mathcal{M}$ satisfies $\forall x\,(P(x) \to Q(x))$, the latter fact forces every element of our model to satisfy Q as well. By combining these two cases (i.e. either all elements of $\mathcal{M}$ satisfy P, or not) we have shown that $\mathcal{M}$ satisfies $\forall x\,P(x) \to \forall x\,Q(x)$.

What about the converse of the above? Is

$$\forall x\,P(x) \to \forall x\,Q(x) \models \forall x\,(P(x) \to Q(x))$$

construct a counter-example model. Let $A' \stackrel{\text{def}}{=} \{a, b\}$, $P^{\mathcal{M}'} \stackrel{\text{def}}{=} \{a\}$ and $Q^{\mathcal{M}'} \stackrel{\text{def}}{=} \{b\}$. Then $\mathcal{M}' \models \forall x\,P(x) \to \forall x\,Q(x)$ holds, but $\mathcal{M}' \models \forall x\,(P(x) \to Q(x))$ does not.

Consider a model $\mathcal{M}$ with $A = \{a, b\}$, $P^{\mathcal{M}} = \{a\}$, $Q^{\mathcal{M}} = \{\}$

T

F because $P(a) \to Q(a)$ is false

UNDECIDABILITY OF PREDICATE LOGIC

Validity in predicate logic. *Given a logical formula ϕ in predicate logic, does $\models \phi$ hold, yes or no?*

Turing Machine - Alan Turing

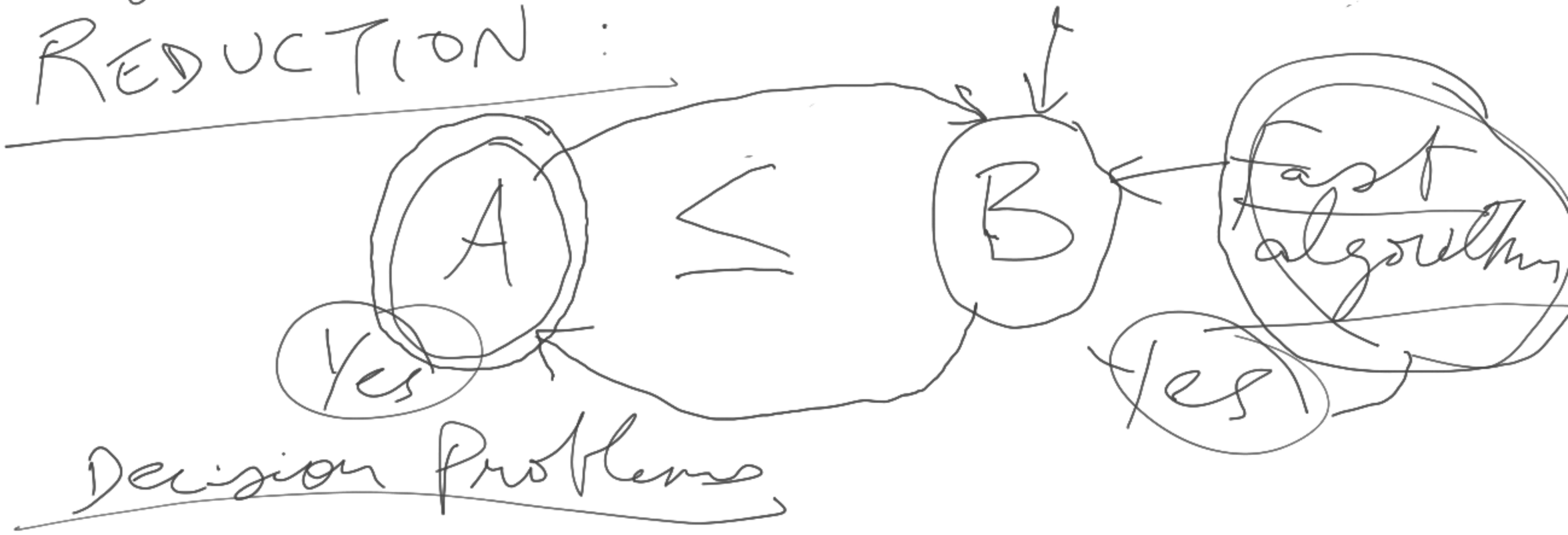
Lambda Calculus - Alonzo Church

→ LISP, Scheme — Functional programming languages

C, Python — Imperative language

Diagonalization argument

REDUCTION:



Decision Problems

known undecidable problem: Post's Correspondence problem

Theorem 2.22 The decision problem of validity in predicate logic is undecidable: no program exists which, given any ϕ, decides whether $\vDash \phi$.

Post's Correspondence Problem $\leq$ validity in predicate logic

Is satisfiability checking decidable? Validity checking $\leq$ Unsatisfiability checking No!

ϕ is unsatisfiable if, and only if, $\neg\phi$ is valid

$\vDash \phi$ iff $\vdash \phi$

Soundness & completeness

Provability is undecidable

EXPRESSIVENESS OF PREDICATE LOGIC OR FIRST-ORDER LOGIC

EXISTENTIAL SECOND-ORDER LOGIC

Example 2.23 Given a set of states $A = \{s_0, s_1, s_2, s_3\}$, let $R^{\mathcal{M}}$ be the set $\{(s_0, s_1), (s_1, s_0), (s_1, s_1), (s_1, s_2), (s_2, s_0), (s_3, s_0), (s_3, s_2)\}$. We may depict this model as a directed graph in Figure 2.5, where an edge (a transition) leads from a node s to a node s' iff $(s, s') \in R^{\mathcal{M}}$. In that case, we often denote this as $s \to s'$.

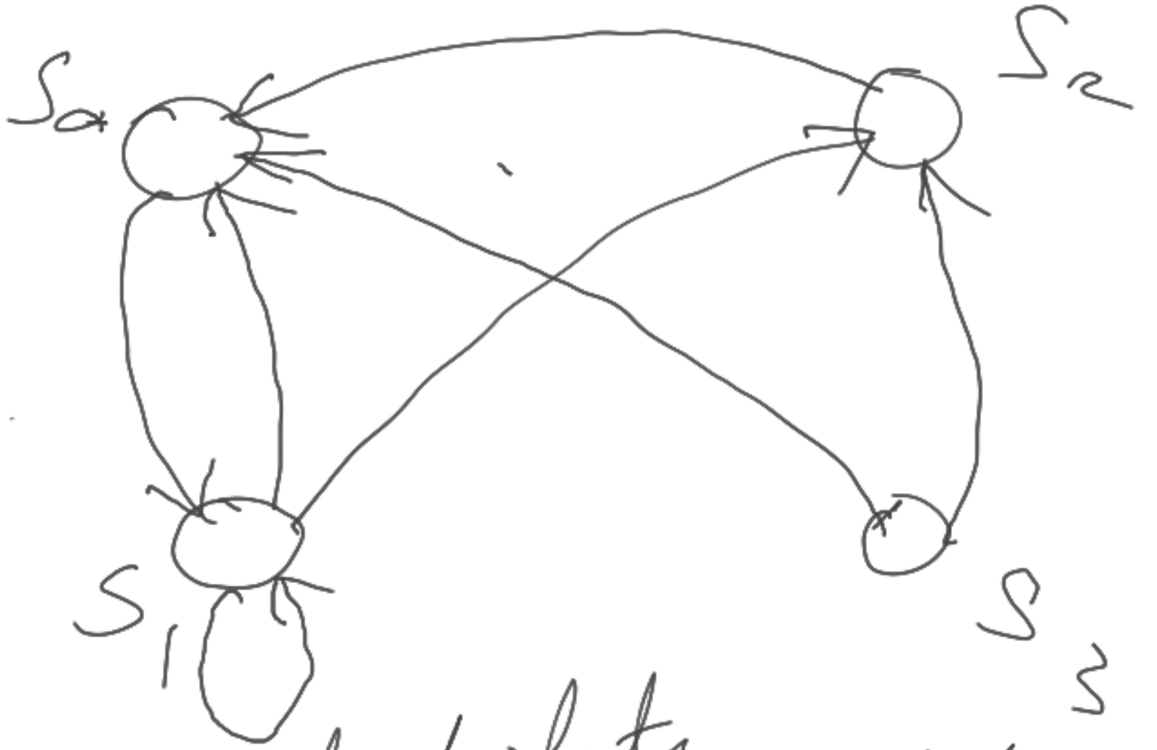
Directed graph

$G(V, E)$

↑ vertex set

$E \subseteq V \times V$

$E = \{(V_1, V_2), (V_1, V_3), (V_3, V_2)\}$

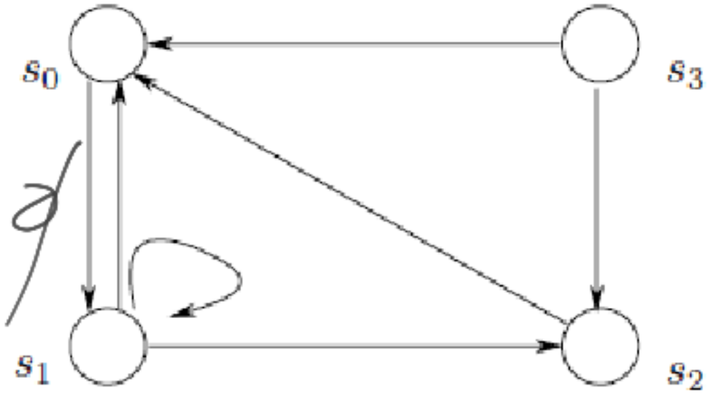


Reachability

e.g. S_2 is reachable from S_0

Is S_3 reachable from S_0? No!

Reachability: Given nodes n and n' in a directed graph, is there a finite path of transitions from n to n'?



Can we find a predicate logic formula ϕ

$$u = v \vee \exists x(R(u,x) \wedge R(x,v)) \vee \exists x_1 \exists x_2(R(u,x_1) \wedge R(x_1,x_2) \wedge R(x_2,v)) \vee \dots$$

with $n \neq n'$ as its only free variable & R as its only predicate symbol such that ϕ holds in directed graphs iff there is a path from n to n'?

Theorem 2.24 (Compactness Theorem) Let Γ be a set of sentences of predicate logic. If all finite subsets of Γ are satisfiable, then so is Γ.

PROOF: We use proof by contradiction: Assume that Γ is not satisfiable. Then the semantic entailment $\Gamma \models \bot$ holds as there is no model in which all $\phi \in \Gamma$ are true. By completeness, this means that the sequent $\Gamma \vdash \bot$ is valid. (Note that this uses a slightly more general notion of sequent in which we may have infinitely many premises at our disposal. Soundness and completeness remain true for that reading.) Thus, this sequent has a proof in natural deduction; this proof – being a finite piece of text – can use only finitely many premises Δ from Γ. But then $\Delta \vdash \bot$ is valid, too, and so $\Delta \models \bot$ follows by soundness. But the latter contradicts the fact that all finite subsets of Γ are consistent. $\square$

Theorem 2.25 (Löwenheim-Skolem Theorem) Let ψ be a sentence of predicate logic such for any natural number $n \geq 1$ there is a model of ψ with at least n elements. Then ψ has a model with infinitely many elements.

PROOF: The formula $\phi_n \stackrel{\text{def}}{=} \exists x_1 \exists x_2 \ldots \exists x_n \bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j)$ specifies that there are at least n elements. Consider the set of sentences $\Gamma \stackrel{\text{def}}{=} \{\psi\} \cup \{\phi_n \mid n \geq 1\}$ and let Δ be any if its finite subsets. Let $k \geq 1$ be such that $n \leq k$ for all n with $\phi_n \in \Delta$. Since the latter set is finite, such a k has to exist. By assumption, $\{\psi, \phi_k\}$ is satisfiable; but $\phi_k \rightarrow \phi_n$ is valid for all $n \leq k$ (why?). Therefore, Δ is satisfiable as well. The compactness theorem then implies that Γ is satisfiable by some model $\mathcal{M}$; in particular, $\mathcal{M} \models \psi$ holds. Since $\mathcal{M}$ satisfies ϕ_n for all $n \geq 1$, it cannot have finitely many elements. $\square$

Theorem 2.25 (Löwenheim-Skolem Theorem) Let ψ be a sentence of predicate logic such for any natural number $n \geq 1$ there is a model of ψ with at least n elements. Then ψ has a model with infinitely many elements.

PROOF: The formula $\phi_n \stackrel{\text{def}}{=} \exists x_1 \exists x_2 \ldots \exists x_n \bigwedge_{1 \leq i < j \leq n} \neg(x_i = x_j)$ specifies that there are at least n elements. Consider the set of sentences $\Gamma \stackrel{\text{def}}{=} \{\psi\} \cup \{\phi_n \mid n \geq 1\}$ and let Δ be any of its finite subsets. Let $k \geq 1$ be such that $n \leq k$ for all n with $\phi_n \in \Delta$. Since the latter set is finite, such a k has to exist. By assumption, $\{\psi, \phi_k\}$ is satisfiable; but $\phi_k \to \phi_n$ is valid for all $n \leq k$ (why?). Therefore, Δ is satisfiable as well. The compactness theorem then implies that Γ is satisfiable by some model $\mathcal{M}$; in particular, $\mathcal{M} \models \psi$ holds. Since $\mathcal{M}$ satisfies ϕ_n for all $n \geq 1$, it cannot have finitely many elements. $\square$

If $\mathcal{M}$ was finitely many elements say q of them, then ϕ_{q+1} should not be satisfiable in $\mathcal{M}$. But it is!

Theorem 2.26 Reachability is not expressible in predicate logic: there is no predicate-logic formula ϕ with u and v as its only free variables and R as its only predicate symbol (of arity 2) such that ϕ holds in directed graphs iff there is a path in that graph from the node associated to u to the node associated to v.

PROOF: Suppose there is a formula ϕ expressing the existence of a path from the node associated to u to the node associated to v. Let c and c' be constants. Let ϕ_n be the formula expressing that there is a path of length n from c to c': we define ϕ_0 as $c = c'$, ϕ_1 as $R(c, c')$ and, for $n > 1$,

$$\phi_n \stackrel{\text{def}}{=} \exists x_1 \ldots \exists x_{n-1}(R(c, x_1) \wedge R(x_1, x_2) \wedge \cdots \wedge R(x_{n-1}, c')).$$

Let $\Delta = \{\neg\phi_i \mid i \geq 0\} \cup \{\phi[c/u][c'/v]\}$. All formulas in Δ are sentences and Δ is unsatisfiable, since the 'conjunction' of all sentences in Δ says that there is no path of length 0, no path of length 1, etc. from the node denoted by c to the node denoted by c', but there is a finite path from c to c' as $\phi[c/u][c'/v]$ is true.

However, every finite subset of Δ is satisfiable since there are paths of any finite length. Therefore, by the Compactness Theorem, Δ itself is satisfiable. This is a contradiction. Therefore, there cannot be such a formula ϕ. □

EXISTENTIAL SECOND-ORDER LOGIC

$$\exists P\, \phi$$

$$\exists P \forall x \forall y \forall z\, (C_1 \wedge C_2 \wedge C_3 \wedge C_4)$$

where each C_i is a Horn clause[4]

$$\begin{aligned} C_1 &\stackrel{\text{def}}{=} P(x, x) \\ C_2 &\stackrel{\text{def}}{=} P(x, y) \wedge P(y, z) \rightarrow P(x, z) \\ C_3 &\stackrel{\text{def}}{=} P(u, v) \rightarrow \bot \\ C_4 &\stackrel{\text{def}}{=} R(x, y) \rightarrow P(x, y). \end{aligned}$$

Given a model $\mathcal{M}$ with interpretations for all function and predicate symbols of ϕ in (2.11), *except* P, let $\mathcal{M}_T$ be that same model augmented with an interpretation $T \subseteq A \times A$ of P, i.e. $P^{\mathcal{M}_T} = T$. For any look-up table l, the semantics of $\exists P\, \phi$ is then

$$\mathcal{M} \vDash_l \exists P\, \phi \qquad \text{iff} \qquad \text{for some } T \subseteq A \times A,\ \mathcal{M}_T \vDash_l \phi. \tag{2.13}$$

UNIVERSAL SECOND-ORDER LOGIC

$$\forall P \, \exists x \exists y \exists z \, (\neg C_1 \vee \neg C_2 \vee \neg C_3 \vee \neg C_4) \qquad (2.14)$$

Theorem 2.28 Let $\mathcal{M} = (A, R^{\mathcal{M}})$ be any model. Then the formula in (2.14) holds under look-up table l in $\mathcal{M}$ iff $l(v)$ is R-reachable from $l(u)$ in $\mathcal{M}$.

Reachability is expressible in Universal Second-order Logic.

Two of the central concepts developed so far are

- *model checking*: given a formula ϕ of predicate logic and a matching model $\mathcal{M}$ determine whether $\mathcal{M} \models \phi$ holds; and
- *semantic entailment*: given a set of formulas Γ of predicate logic, is $\Gamma \models \phi$ valid?

Software Specification & Validation

Γ: Requirements imposed on the S/W design

ϕ: Desired property in any implementation that meets the requirements in Γ

Verification by Model Checking

- safety-critical systems
- commercially critical system

Formal verification of correctness of software systems

Formal verification techniques can be thought of as comprising three parts:

- a *framework for modelling systems*, typically a description language of some sort;
- a *specification language* for describing the properties to be verified;
- a *verification method* to establish whether the description of a system satisfies the specification.

Approaches to verification can be classified according to the following criteria:

Proof-based vs. model-based. In a proof-based approach, the system description is a set of formulas Γ (in a suitable logic) and the specification is another formula ϕ. The verification method consists of trying to find a proof that $\Gamma \vdash \phi$. This typically requires guidance and expertise from the user.

In a model-based approach, the system is represented by a model $\mathcal{M}$ for an appropriate logic. The specification is again represented by a formula ϕ and the verification method consists of computing whether a model $\mathcal{M}$ satisfies ϕ (written $\mathcal{M} \models \phi$). This computation is usually automatic for finite models.

In Chapters 1 and 2, we could see that logical proof systems are often sound and complete, meaning that $\Gamma \vdash \phi$ (provability) holds if, and only if, $\Gamma \models \phi$ (semantic entailment) holds, where the latter is defined as follows: for all models $\mathcal{M}$, if for all $\psi \in \Gamma$ we have $\mathcal{M} \models \psi$, then $\mathcal{M} \models \phi$. Thus, we see that the model-based approach is potentially simpler than the proof-based approach, for it is based on a single model $\mathcal{M}$ rather than a possibly infinite class of them.

Degree of automation. Approaches differ on how automatic the method is; the extremes are fully automatic and fully manual. Many of the computer-assisted techniques are somewhere in the middle.

Full- vs. property-verification. The specification may describe a single property of the system, or it may describe its full behaviour. The latter is typically expensive to verify.

Intended domain of application, which may be hardware or software; sequential or concurrent; reactive or terminating; etc. A reactive system is one which reacts to its environment and is not meant to terminate (e.g., operating systems, embedded systems and computer hardware).

Pre- vs. post-development. Verification is of greater advantage if introduced early in the course of system development, because errors caught earlier in the production cycle are less costly to rectify. (It is alleged that Intel lost millions of dollars by releasing their Pentium chip with the FDIV error.)

TEMPORAL LOGIC

(Models are abstractions!)

- Linear Time Logics (LTL)

think of time as a set of paths where a path is a sequence of time instances

- Branching time Logics (CTL)

represent time as a tree rooted at the present moment & branching out into the future.

Linear-time Temporal Logic (LTL)

SEMANTICS of LTL

Definition 3.4 A transition system $\mathcal{M} = (S, \rightarrow, L)$ is a set of states S endowed with a transition relation $\rightarrow$ (a binary relation on S), such that every $s \in S$ has some $s' \in S$ with $s \rightarrow s'$, and a labelling function $L : S \rightarrow \mathcal{P}(\texttt{Atoms})$.

$\rightarrow$ a Model

NOT considered

$\subseteq S \times S$

(transition relation)

$\rightarrow$ Powerset of the set of Atoms

→ set

Atoms : p, q, r or p_1, p_2, p_3

p : Printer 21 is offline.

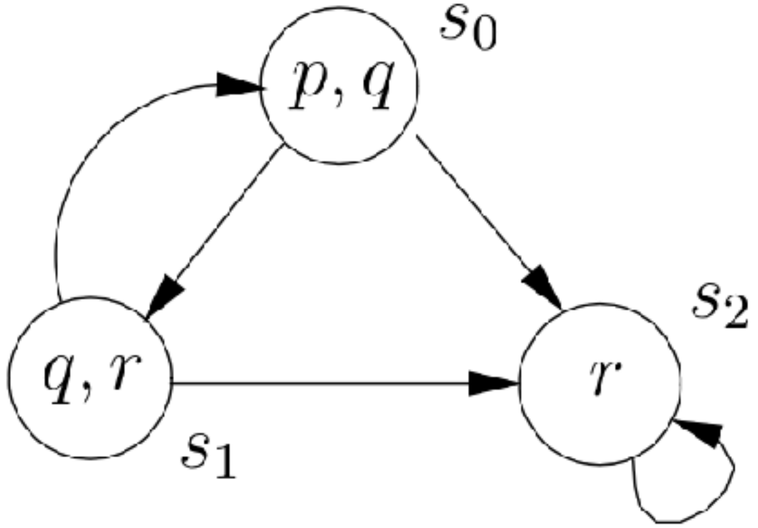
q : Process 1375 is not responding.

Set of Atoms is specified beforehand

our system has only three states s_0, s_1 and s_2; if the only possible transitions between states are $s_0 \to s_1$, $s_0 \to s_2$, $s_1 \to s_0$, $s_1 \to s_2$ and $s_2 \to s_2$; and if $L(s_0) = \{p, q\}$, $L(s_1) = \{q, r\}$ and $L(s_2) = \{r\}$, then we can condense all this information into Figure 3.3. We prefer to present models by means of such pictures whenever that is feasible.

$S = \{s_0, s_1, s_2\}$

Atoms $= \{p, q, r\}$



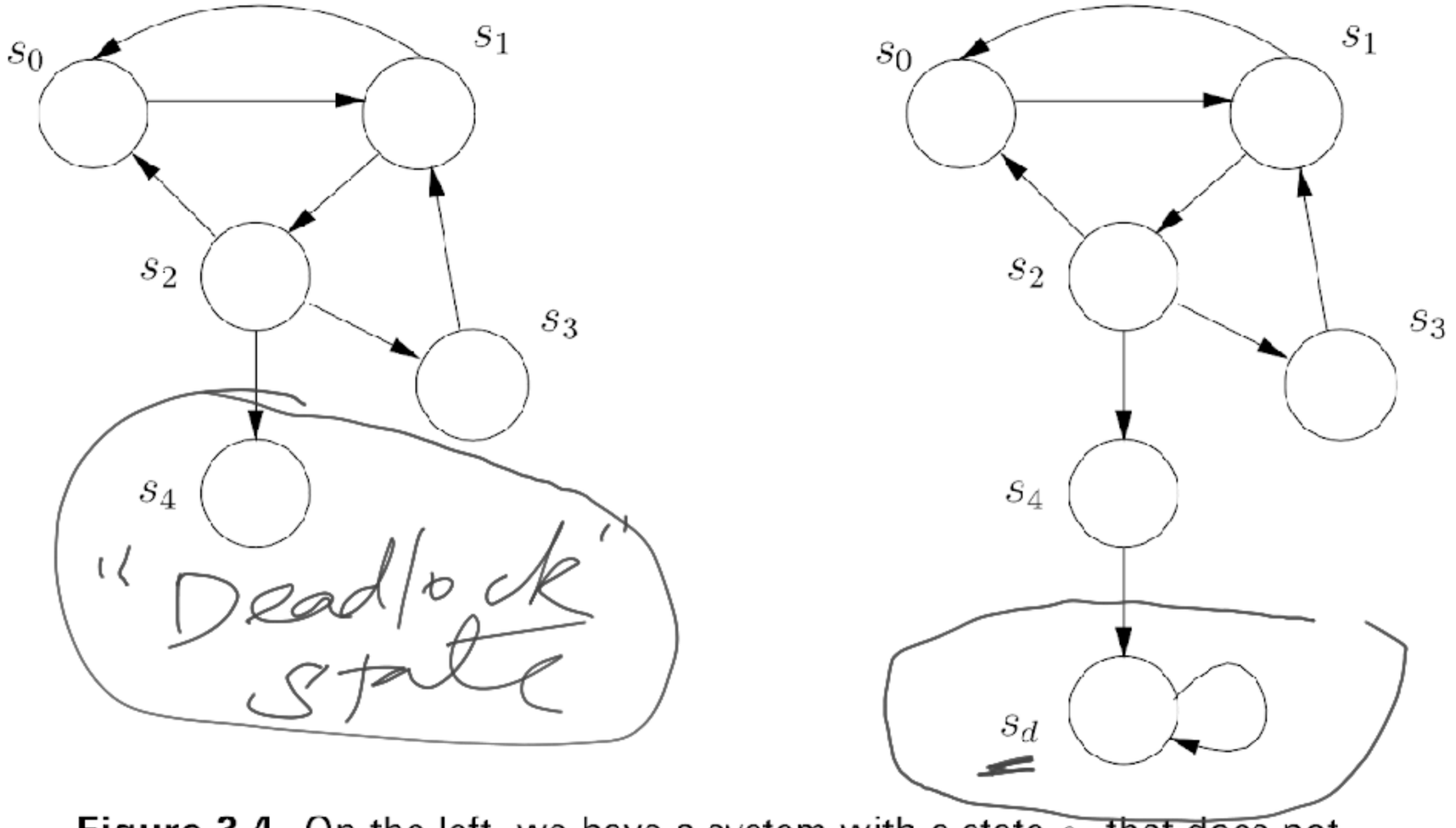


Figure 3.4. On the left, we have a system with a state s_4 that does not have any further transitions. On the right, we expand that system with a 'deadlock' state s_d such that no state can deadlock; of course, it is then our understanding that reaching the 'deadlock' state s_d corresponds to deadlock in the original system.

Definition 3.5 A path in a model $\mathcal{M} = (S, \rightarrow, L)$ is an infinite sequence of states $s_1, s_2, s_3, \ldots$ in S such that, for each $i \geq 1$, $s_i \rightarrow s_{i+1}$. We write the path as $s_1 \rightarrow s_2 \rightarrow \ldots$.

$\pi = s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow s_4 \rightarrow \ldots$

π^i is suffix of π starting at i,

$\pi^3 = s_3 \rightarrow s_4 \rightarrow \ldots$

$\pi^i = s_i \rightarrow s_{i+1} \rightarrow \ldots$

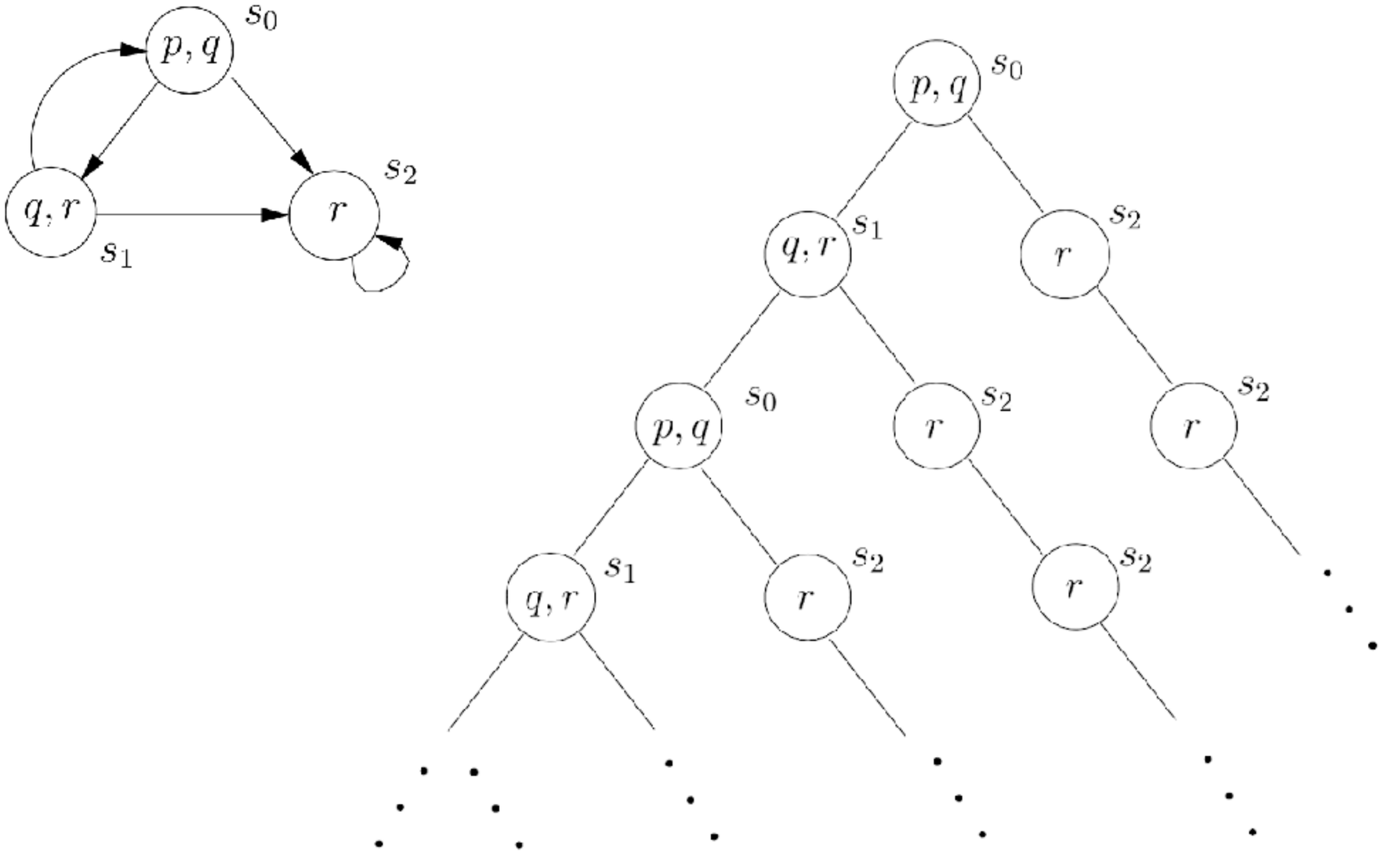


Figure 3.5. Unwinding the system of Figure 3.3 as an infinite tree of all computation paths beginning in a particular state.

SYNTAX

Definition 3.1 Linear-time temporal logic (LTL) has the following syntax given in Backus Naur form:

$$\begin{aligned}\phi ::= &\top \mid \bot \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \\ &\mid (\mathrm{X}\,\phi) \mid (\mathrm{F}\,\phi) \mid (\mathrm{G}\,\phi) \mid (\phi\ \mathrm{U}\ \phi) \mid (\phi\ \mathrm{W}\ \phi) \mid (\phi\ \mathrm{R}\ \phi) \qquad (3.1)\end{aligned}$$

where p is any propositional atom from some set `Atoms`.

Temporal Connectives

X : next state

F : some future state

G : all future states (Globally)

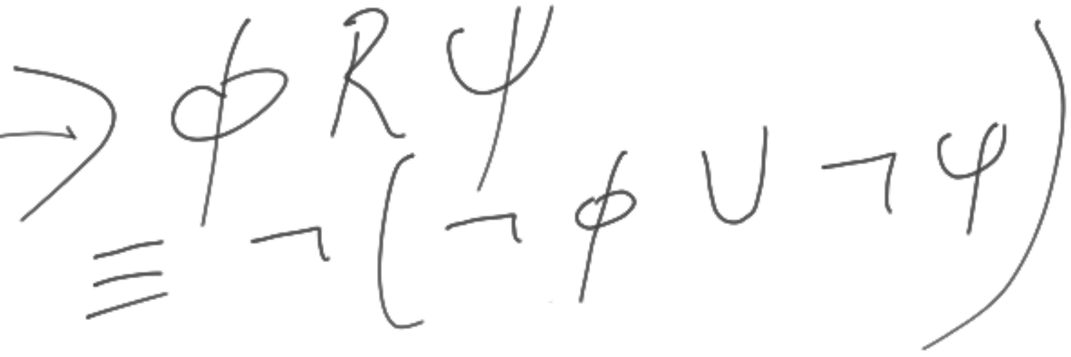
U : Until

R : Release

W : Weak-until

Definition 3.6 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model and $\pi = s_1 \rightarrow \ldots$ be a path in $\mathcal{M}$. Whether π satisfies an LTL formula is defined by the satisfaction relation $\models$ as follows:

1. $\pi \models \top$
2. $\pi \not\models \bot$
3. $\pi \models p$ iff $p \in L(s_1)$
4. $\pi \models \neg\phi$ iff $\pi \not\models \phi$
5. $\pi \models \phi_1 \wedge \phi_2$ iff $\pi \models \phi_1$ and $\pi \models \phi_2$
6. $\pi \models \phi_1 \vee \phi_2$ iff $\pi \models \phi_1$ or $\pi \models \phi_2$
7. $\pi \models \phi_1 \rightarrow \phi_2$ iff $\pi \models \phi_2$ whenever $\pi \models \phi_1$
8. $\pi \models \mathrm{X}\,\phi$ iff $\pi^2 \models \phi$
9. $\pi \models \mathrm{G}\,\phi$ iff, for all $i \geq 1$, $\pi^i \models \phi$
10. $\pi \models \mathrm{F}\,\phi$ iff there is some $i \geq 1$ such that $\pi^i \models \phi$
11. $\pi \models \phi\,\mathrm{U}\,\psi$ iff there is some $i \geq 1$ such that $\pi^i \models \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models \phi$
12. $\pi \models \phi\,\mathrm{W}\,\psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \models \phi$; or for all $k \geq 1$ we have $\pi^k \models \phi$
13. $\pi \models \phi\,\mathrm{R}\,\psi$ iff either there is some $i \geq 1$ such that $\pi^i \models \phi$ and for all $j = 1, \ldots, i$ we have $\pi^j \models \psi$, or for all $k \geq 1$ we have $\pi^k \models \psi$.



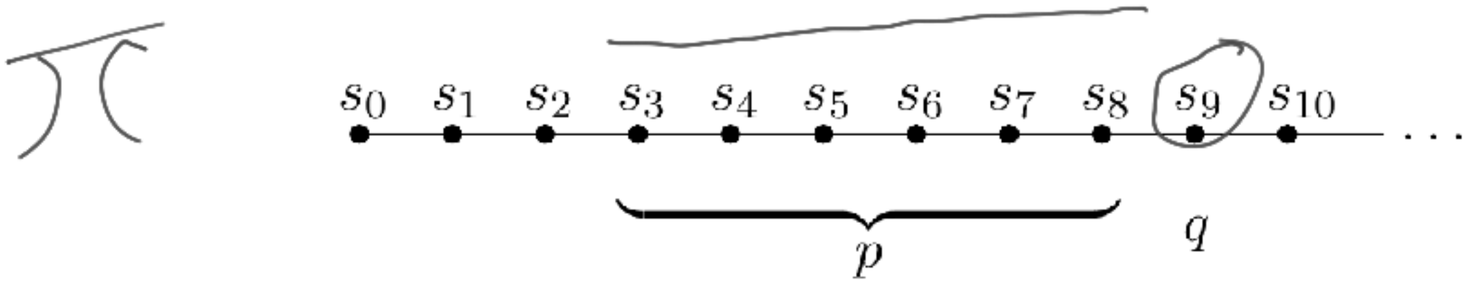


Figure 3.6. An illustration of the meaning of Until in the semantics of LTL. Suppose p is satisfied at (and only at) s_3, s_4, s_5, s_6, s_7, s_8 and q is satisfied at (and only at) s_9. Only the states s_3 to s_9 each satisfy $p \,\mathrm{U}\, q$ along the path shown.

Remark 3.7 Notice that, in clauses 9–13 above, the future includes the present. This means that, when we say 'in all future states,' we are including the present state as a future state. It is a matter of convention whether we do this, or not. As an exercise, you may consider developing a version of LTL in which the future excludes the present. A consequence of adopting the convention that the future shall include the present is that the formulas $\mathrm{G}\, p \to p$, $p \to q\ \mathrm{U}\ p$ and $p \to \mathrm{F}\, p$ are true in every state of every model.

Definition 3.8 Suppose $\mathcal{M} = (S, \to, L)$ is a model, $s \in S$, and ϕ an LTL formula. We write $\mathcal{M}, s \vDash \phi$ if, for every execution path π of $\mathcal{M}$ starting at s, we have $\pi \vDash \phi$.

If $\mathcal{M}$ is clear from the context, we may abbreviate $\mathcal{M}, s \vDash \phi$ by $s \vDash \phi$.

Here are some examples of LTL formulas:

- $((\mathrm{F}\, p) \wedge (\mathrm{G}\, q)) \rightarrow (p\ \mathrm{W}\ r))$
- $(\mathrm{F}\, (p \rightarrow (\mathrm{G}\, r)) \vee ((\neg q)\ \mathrm{U}\ p))$, the parse tree of this formula is illustrated in Figure 3.1.
- $(p\ \mathrm{W}\ (q\ \mathrm{W}\ r))$
- $((\mathrm{G}\, (\mathrm{F}\, p)) \rightarrow (\mathrm{F}\, (q \vee s)))$.

BINDING PRIORITIES

Convention 3.2 The unary connectives (consisting of $\neg$ and the temporal connectives X, F and G) bind most tightly. Next in the order come U, R and W; then come $\wedge$ and $\vee$; and after that comes $\rightarrow$.

- $(\mathrm{F}\, p \wedge \mathrm{G}\, q) \rightarrow (p\ \mathrm{W}\ r)$
- $\mathrm{F}\, (p \rightarrow \mathrm{G}\, r) \vee \neg q\ \mathrm{U}\ p$
- $p\ \mathrm{W}\ (q\ \mathrm{W}\ r)$
- $\mathrm{G}\, \mathrm{F}\, p \rightarrow \mathrm{F}\, (q \vee s)$.

The following are *not* well-formed formulas:

- $\mathrm{U}\, r$ – since U is binary, not unary
- $p\ \mathrm{G}\ q$ – since G is unary, not binary.

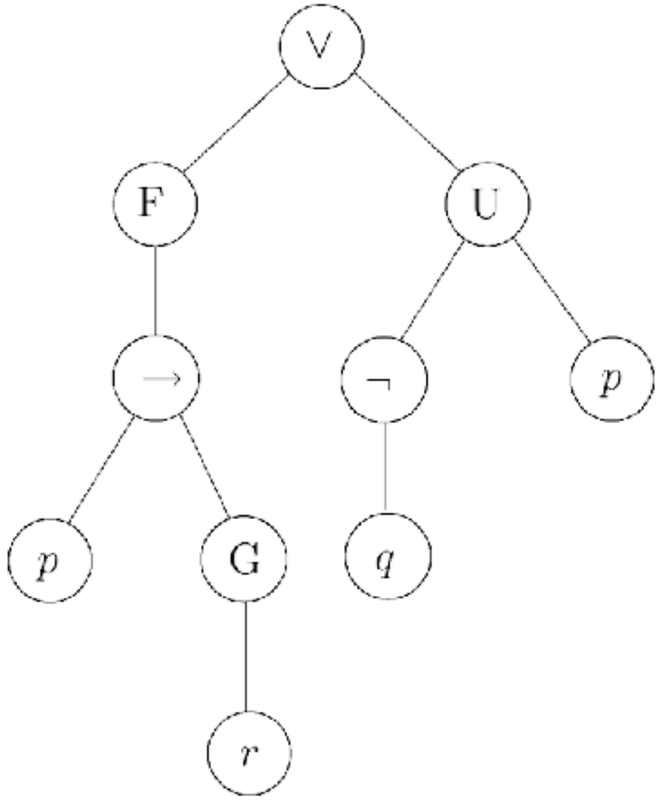


Figure 3.1. The parse tree of $(\mathrm{F}\,(p \rightarrow \mathrm{G}\,r) \vee ((\neg q)\ \mathrm{U}\ p))$.

Definition 3.3 A subformula of an LTL formula ϕ is any formula ψ whose parse tree is a subtree of ϕ's parse tree.

The subformulas of $p\ \mathrm{W}\ (q\ \mathrm{U}\ r)$, e.g., are p, q, r, $q\ \mathrm{U}\ r$ and $p\ \mathrm{W}\ (q\ \mathrm{U}\ r)$.

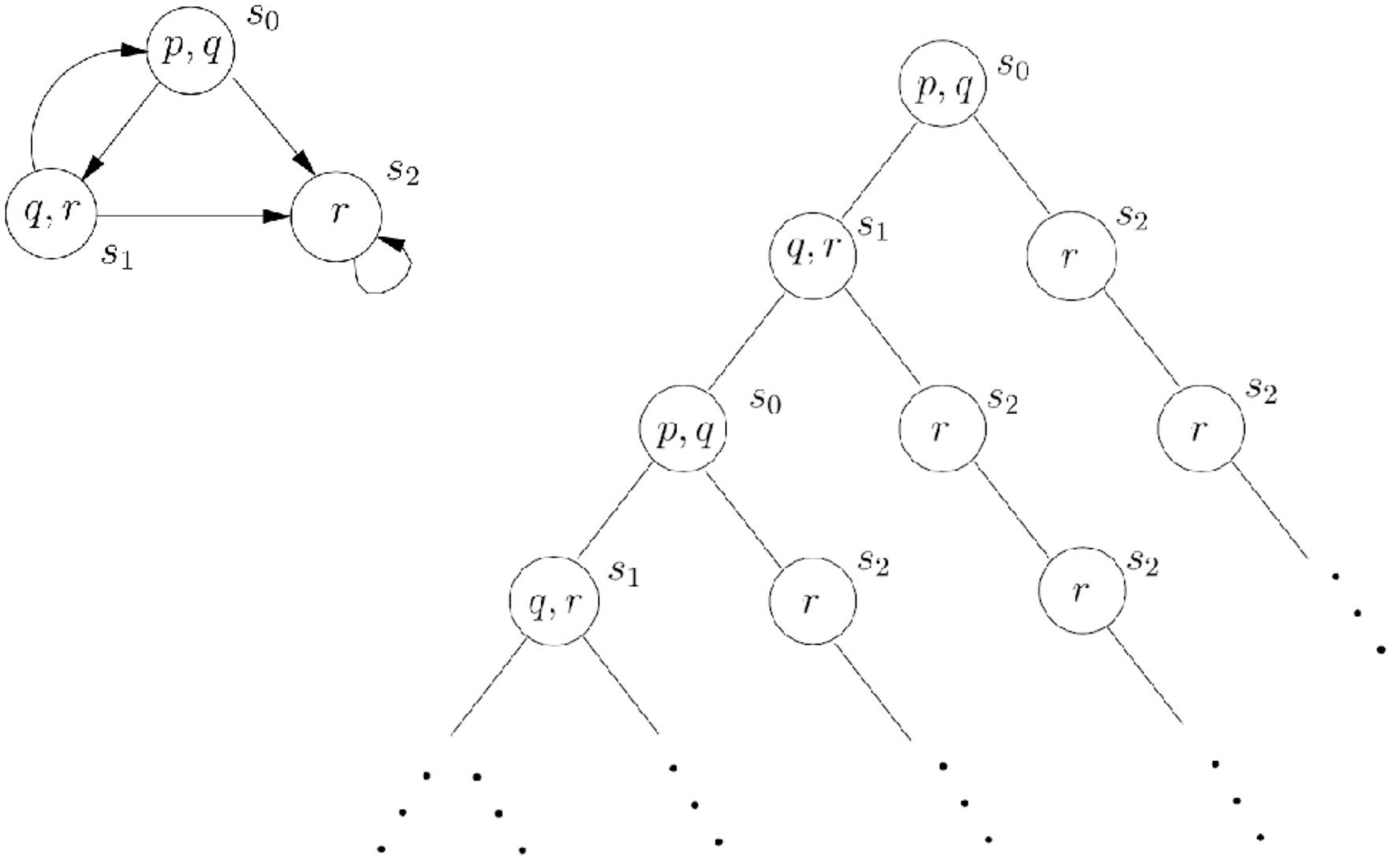


Figure 3.5. Unwinding the system of Figure 3.3 as an infinite tree of all computation paths beginning in a particular state.

EXAMPLES

1. $\mathcal{M}, s_0 \models p \wedge q$ holds since the atomic symbols p and q are contained in the node of s_0: $\pi \models p \wedge q$ for *every* path π beginning in s_0.
2. $\mathcal{M}, s_0 \models \neg r$ holds since the atomic symbol r is *not* contained in node s_0.
3. $\mathcal{M}, s_0 \models \top$ holds by definition.
4. $\mathcal{M}, s_0 \models \mathrm{X}\, r$ holds since all paths from s_0 have either s_1 or s_2 as their next state, and each of those states satisfies r.
5. $\mathcal{M}, s_0 \models \mathrm{X}\,(q \wedge r)$ does not hold since we have the rightmost computation path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ in Figure 3.5, whose second node s_2 contains r, but not q.
6. $\mathcal{M}, s_0 \models \mathrm{G}\,\neg(p \wedge r)$ holds since all computation paths beginning in s_0 satisfy $\mathrm{G}\,\neg(p \wedge r)$, i.e. they satisfy $\neg(p \wedge r)$ in each state along the path. Notice that $\mathrm{G}\,\phi$ holds in a state if, and only if, ϕ holds in all states reachable from the given state.
7. For similar reasons, $\mathcal{M}, s_2 \models \mathrm{G}\, r$ holds (note the s_2 instead of s_0).
8. For any state s of $\mathcal{M}$, we have $\mathcal{M}, s \models \mathrm{F}\,(\neg q \wedge r) \to \mathrm{F}\,\mathrm{G}\, r$. This says that if any path π beginning in s gets to a state satisfying $(\neg q \wedge r)$, then the path π satisfies $\mathrm{F}\,\mathrm{G}\, r$. Indeed this is true, since if the path has a state satisfying $(\neg q \wedge r)$ then (since that state must be s_2) the path does satisfy $\mathrm{F}\,\mathrm{G}\, r$. Notice what $\mathrm{F}\,\mathrm{G}\, r$ says about a path: eventually, you have continuously r.

9. The formula $\mathrm{G}\,\mathrm{F}\,p$ expresses that p occurs along the path in question infinitely often. Intuitively, it's saying: no matter how far along the path you go (that's the G part) you will find you still have a p in front of you (that's the F part). For example, the path $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$ satisfies $\mathrm{G}\,\mathrm{F}\,p$. But the path $s_0 \to s_2 \to s_2 \to s_2 \to \ldots$ doesn't.
10. In our model, if a path from s_0 has infinitely many ps on it then it must be the path $s_0 \to s_1 \to s_0 \to s_1 \to \ldots$, and in that case it also has infinitely many rs on it. So, $\mathcal{M}, s_0 \vDash \mathrm{G}\,\mathrm{F}\,p \to \mathrm{G}\,\mathrm{F}\,r$. But it is not the case the other way around! It is not the case that $\mathcal{M}, s_0 \vDash \mathrm{G}\,\mathrm{F}\,r \to \mathrm{G}\,\mathrm{F}\,p$, because we can find a path from s_0 which has infinitely many rs but only one p.

Definition 3.9 We say that two LTL formulas ϕ and ψ are semantically equivalent, or simply equivalent, writing $\phi \equiv \psi$, if for all models $\mathcal{M}$ and all paths π in $\mathcal{M}$: $\pi \models \phi$ iff $\pi \models \psi$.

In propositional logic, we saw that $\wedge$ and $\vee$ are duals of each other, ing that if you push a $\neg$ past a $\wedge$, it becomes a $\vee$, and vice versa:

$$\neg(\phi \wedge \psi) \equiv \neg\phi \vee \neg\psi \qquad \neg(\phi \vee \psi) \equiv \neg\phi \wedge \neg\psi.$$

Similarly, F and G are duals of each other, and X is dual with itself:

$$\neg \mathrm{G}\, \phi \equiv \mathrm{F}\, \neg\phi \qquad \neg \mathrm{F}\, \phi \equiv \mathrm{G}\, \neg\phi \qquad \neg \mathrm{X}\, \phi \equiv \mathrm{X}\, \neg\phi.$$

Also U and R are duals of each other:

$$\neg(\phi\ \mathrm{U}\ \psi) \equiv \neg\phi\ \mathrm{R}\ \neg\psi \qquad \neg(\phi\ \mathrm{R}\ \psi) \equiv \neg\phi\ \mathrm{U}\ \neg\psi.$$

READING: sec 3.2.3

Examples of things LTL can't do

- From ~~any~~ a state *it is possible* to get to a **restart** state (i.e., there is a path from all states to a state satisfying **restart**).
- The lift *can* remain idle on the third floor with its doors closed (i.e., from the state in which it is on the third floor, there is a path along which it stays there).

LTL Equivalence (cont'd..)

It's also the case that F distributes over $\vee$ and G over $\wedge$, i.e.,

$$\mathrm{F}\,(\phi \vee \psi) \equiv \mathrm{F}\,\phi \vee \mathrm{F}\,\psi$$
$$\mathrm{G}\,(\phi \wedge \psi) \equiv \mathrm{G}\,\phi \wedge \mathrm{G}\,\psi.$$

→ F does not distribute over $\wedge$!

$$\mathrm{F}\,\phi \equiv \top\ \mathrm{U}\ \phi$$

$$\phi\ \mathrm{U}\ \psi \equiv (\phi\ \mathrm{W}\ \psi) \wedge (\mathrm{F}\,\psi\,. \tag{3.2}$$

Proof (of one side):

To prove equivalence (3.2), suppose first that a path satisfies $\phi\ \mathrm{U}\ \psi$. Then, from clause 11, we have $i \geq 1$ such that $\pi^i \vDash \psi$ and for all $j = 1, \ldots, i-1$ we have $\pi^j \vDash \phi$. From clause 12, this proves $\phi\ \mathrm{W}\ \psi$, and from clause 10 it proves $\mathrm{F}\,\psi$. Thus for all paths π, if $\pi \vDash \phi\ \mathrm{U}\ \psi$ then $\pi \vDash \phi\ \mathrm{W}\ \psi \wedge \mathrm{F}\,\psi$. As an exercise, the reader can prove it the other way around.

Prove that if $\phi\ \mathrm{U}\ \psi$ is true, then $\phi\ \mathrm{W}\ \psi \wedge \mathrm{F}\,\psi$ is true.

ADEQUATE SETS OF CONNECTIVES in LTL

Small adequate sets of connectives also exist in LTL. Here is a summary of the situation.

- X is completely orthogonal to the other connectives. That is to say, its presence doesn't help in defining any of the other ones in terms of each other. Moreover, X cannot be derived from any combination of the others.
- Each of the sets {U, X}, {R, X}, {W, X} is adequate.

Recall: in Prop Logic $\{\bot, \wedge, \neg, \rightarrow\}$ is an adequate set of connectives since $\wedge, \rightarrow, \top$ can be written using them.

Branching-Time Logic

CTL

A : along All paths

E : along at least one path (there Exists)

Computation Tree Logic, or CTL for short, is a *branching-time* logic, meaning that its model of time is a tree-like structure in which the future is not determined; there are different paths in the future, any one of which might be the 'actual' path that is realised.

Syntax

Definition 3.12 We define CTL formulas inductively via a Backus Naur form as done for LTL:

$$\phi ::= \bot \mid \top \mid p \mid (\neg\phi) \mid (\phi \wedge \phi) \mid (\phi \vee \phi) \mid (\phi \rightarrow \phi) \mid \mathrm{AX}\,\phi \mid \mathrm{EX}\,\phi \mid \mathrm{AF}\,\phi \mid \mathrm{EF}\,\phi \mid \mathrm{AG}\,\phi \mid \mathrm{EG}\,\phi \mid \mathrm{A}[\phi\ \mathrm{U}\ \phi] \mid \mathrm{E}[\phi\ \mathrm{U}\ \phi]$$

where p ranges over a set of atomic formulas.

Symbol pairs are indivisible!

CTL does not subsume LTL

Convention 3.13 We assume similar binding priorities for the CTL connectives to what we did for propositional and predicate logic. The unary connectives (consisting of $\neg$ and the temporal connectives AG, EG, AF, EF, AX and EX) bind most tightly. Next in the order come $\wedge$ and $\vee$; and after that come $\rightarrow$, AU and EU.

$$\mathrm{A}[\mathrm{AX}\,\neg p\ \mathrm{U}\ \mathrm{E}[\mathrm{EX}\,(p \wedge q)\ \mathrm{U}\ \neg p]]$$

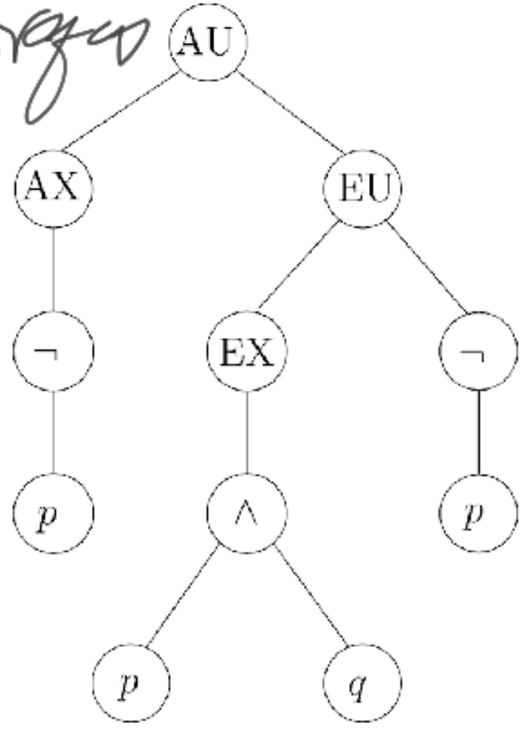


Figure 3.18. The parse tree of a CTL formula without infix notation.

Definition 3.14 A subformula of a CTL formula ϕ is any formula ψ whose parse tree is a subtree of ϕ's parse tree.

SEMANTICS OF CTL

Informally

CTL formulas are interpreted over transition systems (Definition 3.4). Let $\mathcal{M} = (S, \rightarrow, L)$ be such a model, $s \in S$ and ϕ a CTL formula. The definition of whether $\mathcal{M}, s \models \phi$ holds is recursive on the structure of ϕ, and can be roughly understood as follows:

- If ϕ is atomic, satisfaction is determined by L.
- If the top-level connective of ϕ (i.e., the connective occurring top-most in the parse tree of ϕ) is a boolean connective ($\wedge$, $\vee$, $\neg$, $\top$ etc.) then the satisfaction question is answered by the usual truth-table definition and further recursion down ϕ.
- If the top level connective is an operator beginning A, then satisfaction holds if all paths from s satisfy the 'LTL formula' resulting from removing the A symbol.
- Similarly, if the top level connective begins with E, then satisfaction holds if some path from s satisfy the 'LTL formula' resulting from removing the E.

In the last two cases, the result of removing A or E is not strictly an LTL formula, for it may contain further As or Es below. However, these will be dealt with by the recursion.

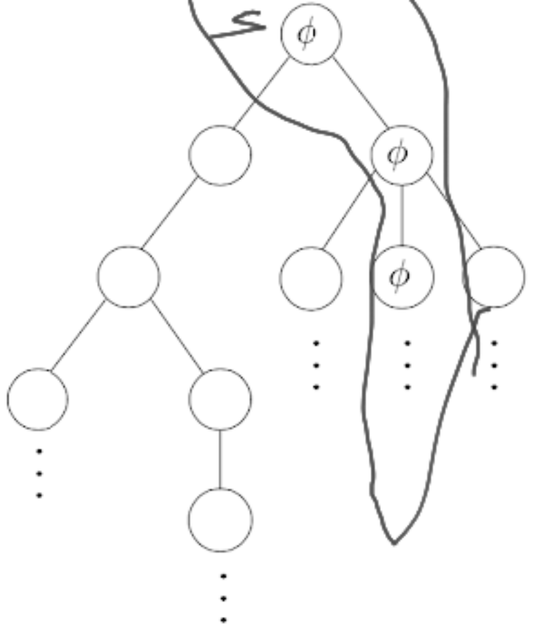
Formal definitions

Definition 3.15 Let $\mathcal{M} = (S, \rightarrow, L)$ be a model for CTL, s in S, ϕ a CTL formula. The relation $\mathcal{M}, s \models \phi$ is defined by structural induction on ϕ:

1. $\mathcal{M}, s \models \top$ and $\mathcal{M}, s \not\models \bot$
2. $\mathcal{M}, s \models p$ iff $p \in L(s)$
3. $\mathcal{M}, s \models \neg\phi$ iff $\mathcal{M}, s \not\models \phi$
4. $\mathcal{M}, s \models \phi_1 \wedge \phi_2$ iff $\mathcal{M}, s \models \phi_1$ and $\mathcal{M}, s \models \phi_2$
5. $\mathcal{M}, s \models \phi_1 \vee \phi_2$ iff $\mathcal{M}, s \models \phi_1$ or $\mathcal{M}, s \models \phi_2$
6. $\mathcal{M}, s \models \phi_1 \rightarrow \phi_2$ iff $\mathcal{M}, s \not\models \phi_1$ or $\mathcal{M}, s \models \phi_2$.
7. $\mathcal{M}, s \models \mathrm{AX}\,\phi$ iff for all s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus, AX says: 'in every next state.'
8. $\mathcal{M}, s \models \mathrm{EX}\,\phi$ iff for some s_1 such that $s \rightarrow s_1$ we have $\mathcal{M}, s_1 \models \phi$. Thus, EX says: 'in some next state.' E is dual to A – in exactly the same way that $\exists$ is dual to $\forall$ in predicate logic.
9. $\mathcal{M}, s \models \mathrm{AG}\,\phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and all s_i along the path, we have $\mathcal{M}, s_i \models \phi$. Mnemonically: for All computation paths beginning in s the property ϕ holds Globally. Note that 'along the path' includes the path's initial state s.
10. $\mathcal{M}, s \models \mathrm{EG}\,\phi$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for all s_i along the path, we have $\mathcal{M}, s_i \models \phi$. Mnemonically: there Exists a path beginning in s such that ϕ holds Globally along the path.

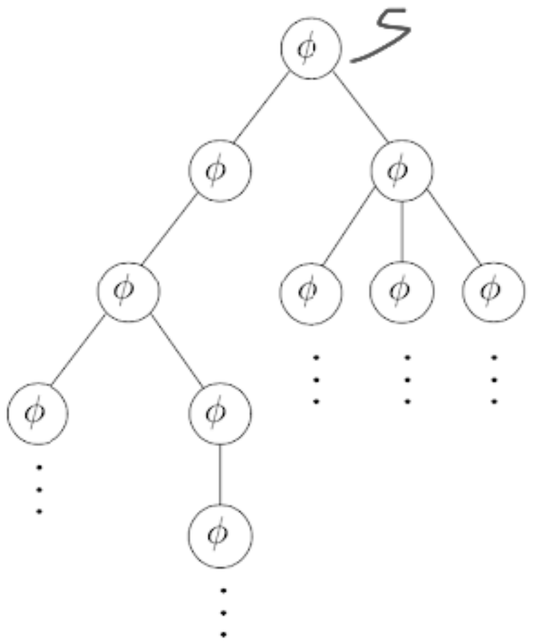
11. $\mathcal{M}, s \vDash \mathrm{AF}\,\phi$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow \ldots$, where s_1 equals s, there is some s_i such that $\mathcal{M}, s_i \vDash \phi$. Mnemonically: for All computation paths beginning in s there will be some Future state where ϕ holds.
12. $\mathcal{M}, s \vDash \mathrm{EF}\,\phi$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and for some s_i along the path, we have $\mathcal{M}, s_i \vDash \phi$. Mnemonically: there Exists a computation path beginning in s such that ϕ holds in some Future state;
13. $\mathcal{M}, s \vDash \mathrm{A}[\phi_1\ \mathrm{U}\ \phi_2]$ holds iff for all paths $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, that path satisfies $\phi_1\ \mathrm{U}\ \phi_2$, i.e., there is some s_i along the path, such that $\mathcal{M}, s_i \vDash \phi_2$, and, for each $j < i$, we have $\mathcal{M}, s_j \vDash \phi_1$. Mnemonically: All computation paths beginning in s satisfy that ϕ_1 Until ϕ_2 holds on it.
14. $\mathcal{M}, s \vDash \mathrm{E}[\phi_1\ \mathrm{U}\ \phi_2]$ holds iff there is a path $s_1 \rightarrow s_2 \rightarrow s_3 \rightarrow \ldots$, where s_1 equals s, and that path satisfies $\phi_1\ \mathrm{U}\ \phi_2$ as specified in 13. Mnemonically: there Exists a computation path beginning in s such that ϕ_1 Until ϕ_2 holds on it.

Figure 3.19. A system whose starting state satisfies $\mathrm{EF}\ \phi$.



$M, s \models EG\phi$

Figure 3.20. A system whose starting state satisfies $\mathrm{EG}\ \phi$.



$M, s \models AG\phi$

Figure 3.21. A system whose starting state satisfies $\mathrm{AG}\ \phi$.

EQUIVALENCES

BET.

CTL formulas

A : universal quantifier over paths

E : existential

G : Universal quantifier on states along a particular path

F : Existential

DE MORGAN'S RULES

Definition 3.16 Two CTL formulas ϕ and ψ are said to be semantically equivalent if any state in any model which satisfies one of them also satisfies the other; we denote this by $\phi \equiv \psi$.

$$\neg \mathrm{AF}\, \phi \equiv \mathrm{EG}\, \neg \phi$$

$$\neg \mathrm{EF}\, \phi \equiv \mathrm{AG}\, \neg \phi$$

$$\neg \mathrm{AX}\, \phi \equiv \mathrm{EX}\, \neg \phi.$$

$$\mathrm{AF}\,\phi \;\equiv\; \mathrm{A}[\top\ \mathrm{U}\ \phi] \qquad \mathrm{EF}\,\phi \;\equiv\; \mathrm{E}[\top\ \mathrm{U}\ \phi]$$

ADEQUATE SET OF CONNECTIVES IN CTL

Theorem 3.17 A set of temporal connectives in CTL is adequate if, and only if, it contains at least one of $\{\mathrm{AX}, \mathrm{EX}\}$, at least one of $\{\mathrm{EG}, \mathrm{AF}, \mathrm{AU}\}$ and EU.

EXPRESSIVE POWERS OF CTL & LTL

Things you can express in LTL but not in CTL

→ LTL formula $\phi : Fp \rightarrow Fq$

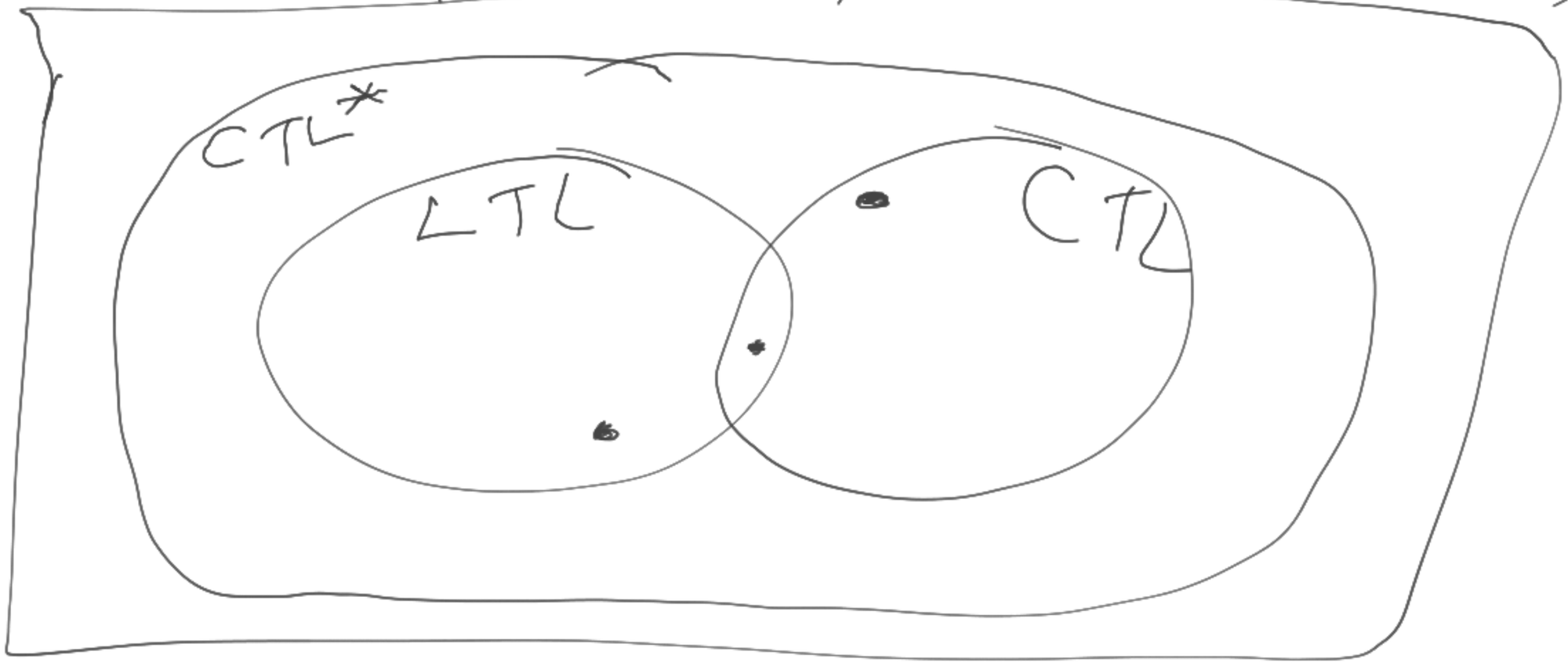
$s \models \phi$

cannot write in CTL

What about $AFp \rightarrow AFq$?

Venn Diagram

(Expressivity of LTL & CTL)



PROGRAM VERIFICATION

Types of verification that we will study:

Proof-based. We do not exhaustively check every state that the system can get in to, as one does with model checking; this would be impossible, given that program variables can have infinitely many interacting values. Instead, we construct a proof that the system satisfies the property at hand, using a proof calculus. This is analogous to the situation in Chapter 2, where using a suitable proof calculus avoided the problem of having to check infinitely many models of a set of predicate logic formulas in order to establish the validity of a sequent.

Semi-automatic. Although many of the steps involved in proving that a program satisfies its specification are mechanical, there are some steps that involve some intelligence and that cannot be carried out algorithmically by a computer. As we will see, there are often good heuristics to help the programmer complete these tasks. This contrasts with the situation of the last chapter, which was fully automatic.

Property-oriented. Just like in the previous chapter, we verify properties of a program rather than a full specification of its behaviour.

Application domain. The domain of application in this chapter is sequential transformational programs. 'Sequential' means that we assume the program runs on a single processor and that there are no concurrency issues. 'Transformational' means that the program takes an input and, after some computation, is expected to terminate with an output. For example, methods of objects in Java are often programmed in this style. This contrasts with the previous chapter which focuses on reactive systems that are not intended to terminate and that react continually with their environment.

Pre/post-development. The techniques of this chapter should be used during the coding process for small fragments of program that perform an identifiable (and hence, specifiable) task and hence should be used during the development process in order to avoid functional bugs.

Reading: Sec 4.1

Framework for Software Verification

- Convert the informal description R of requirements for an application domain into an 'equivalent' formula ϕ_R of some symbolic logic;
- Write a program P which is meant to realise ϕ_R in the programming environment supplied by your company, or wanted by the particular customer;
- *Prove* that the program P satisfies the formula ϕ_R.

A first approximation:
Does not cover basic aspects
talking to customer, modelling
a specification etc.

CORE LANGUAGE

Selling is a C-like language without objects, procedures, threads, recursion/recursive data str

3 syntactic domains

- Integer Expressions
- Boolean expression if
- Commands

Grammar for integer expressions

$$E ::= n \mid x \mid (-E) \mid (E + E) \mid (E - E) \mid (E * E)$$

where n is any numeral in $\{\ldots, -2, -1, 0, 1, 2, \ldots\}$ and x is any variable.

unary minus

Convention 4.1 In the grammar above, ~~negation~~ $-$ binds more tightly than multiplication $*$, which binds more tightly than subtraction $-$ and addition $+$.

Grammar for Boolean Expressions

$$B ::= \texttt{true} \mid \texttt{false} \mid (!B) \mid (B \,\&\, B) \mid (B \,||\, B) \mid (E < E)$$

negation | disjunction — less than

conjunction

Text: how to make equality

Grammar for Commands

$$C \;::=\; \mathtt{x} = E \;\mid\; C;C \;\mid\; \texttt{if}\ B\ \{C\}\ \texttt{else}\ \{C\} \;\mid\; \texttt{while}\ B\ \{C\}$$

Intuitive Meaning

1. The atomic command $\mathtt{x} = E$ is the usual assignment statement; it evaluates the integer expression E in the current state of the store and then overwrites the current value stored in x with the result of that evaluation.
2. The compound command $C_1; C_2$ is the sequential composition of the commands C_1 and C_2. It begins by executing C_1 in the current state of the store. If that execution terminates, then it executes C_2 in the storage state resulting from the execution of C_1. Otherwise – if the execution of C_1 does not terminate – the run of $C_1; C_2$ also does not terminate. Sequential composition is an example of a *control structure* since it implements a certain policy of flow of control in a computation.
3. Another control structure is $\texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}$. It first evaluates the boolean expression B in the current state of the store; if that result is true, then C_1 is executed; if B evaluated to false, then C_2 is executed.
4. The third control construct $\texttt{while}\ B\ \{C\}$ allows us to write statements which are executed repeatedly. Its meaning is that:

   a the boolean expression B is evaluated in the current state of the store;
   b if B evaluates to false, then the command terminates,
   c otherwise, the command C will be executed. If that execution terminates, then we resume at step (a) with a re-evaluation of B as the updated state of the store may have changed its value.

```
y = 1;
z = 0;
while (z != x) {
    z = z + 1;
    y = y * z;
}
```

Compute a number y whose square is less than the input x.

If the input x is a positive number, compute a number whose square is less than x.

$$\{\!|\phi|\!\}\ P\ \{\!|\psi|\!\} \tag{4.5}$$

which (roughly) means:

> If the program P is run in a state that satisfies ϕ, then the state resulting from P's execution will satisfy ψ.

The specification of the program P, to calculate a number whose square is less than x, now looks like this:

$$\{\!|x > 0|\!\}\ P\ \{\!|y \cdot y < x|\!\}. \tag{4.6}$$

Definition 4.3 1. The form $(\!|\phi|\!)\, P\, (\!|\psi|\!)$ of our specification is called a Hoare triple, after the computer scientist C. A. R. Hoare.

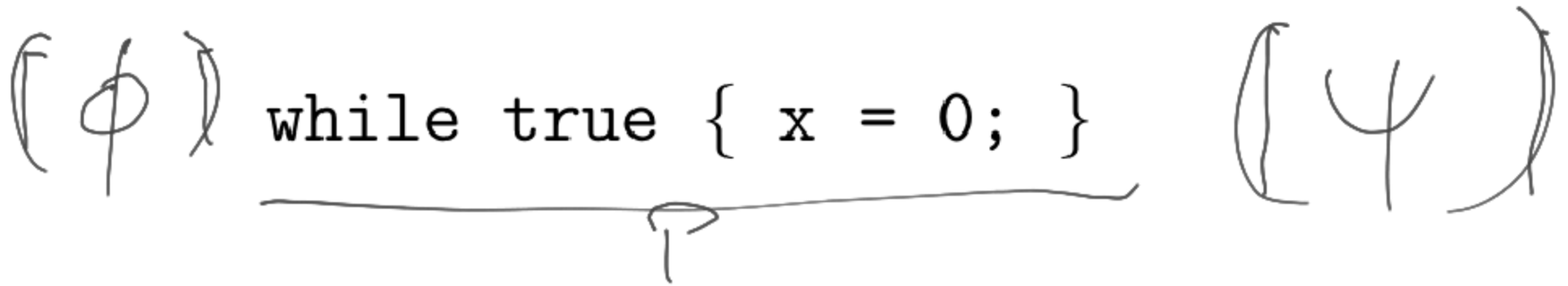
2. In (4.5), the formula ϕ is called the precondition of P and ψ is called the postcondition.
3. A store or state of core programs is a function l that assigns to each variable x an integer $l(x)$.
4. For a formula ϕ of predicate logic with function symbols $-$ (unary), $+$, $-$, and $*$ (binary); and a binary predicate symbols $<$ and $=$, we say that a state l satisfies ϕ or l is a ϕ-state – written $l \models \phi$ – iff $\mathcal{M} \models_l \phi$ from page 128 holds, where l is viewed as a look-up table and the model $\mathcal{M}$ has as set A all integers and interprets the function and predicate symbols in their standard manner.
5. For Hoare triples in (4.5), we demand that quantifiers in ϕ and ψ only bind variables that do not occur in the program P.

Example 4.4 For any state l for which $l(x) = -2$, $l(y) = 5$, and $l(z) = -1$, the relation

1. $l \models \neg(x + y < z)$ holds since $x + y$ evaluates to $-2 + 5 = 3$, z evaluates to $l(z) = -1$, and 3 is not strictly less than -1;

$(\!|\top|\!)\, P\, (\!|\psi|\!)$ We set precondition to T if there is no precondition

Definition 4.5 (Partial correctness) We say that the triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is satisfied under partial correctness if, for all states which satisfy ϕ, the state resulting from P's execution satisfies the postcondition ψ, provided that P actually terminates. In this case, the relation $\vDash_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ holds. We call $\vDash_{\mathsf{par}}$ the satisfaction relation for partial correctness.



Definition 4.6 (Total correctness) We say that the triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ is satisfied under total correctness if, for all states in which P is executed which satisfy the precondition ϕ, P is guaranteed to terminate and the resulting state satisfies the postcondition ψ. In this case, we say that $\vDash_{\mathsf{tot}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ holds and call $\vDash_{\mathsf{tot}}$ the satisfaction relation of total correctness.

EXAMPLE OF A FORMAL SPEC

Input: x

1. Let Succ be the program

```
a = x + 1;
if (a - 1 == 0) {
    y = 1;
} else {
    y = a;
}
```

The program Succ satisfies the specification $(\!|\top|\!)\ \texttt{Succ}\ (\!|y = (x + 1)|\!)$ under partial and total correctness, so if we think of x as input and y as output, then Succ computes the successor function. Note that this code is far from optimal.

Definition 4.8 1. If the partial correctness of triples $(\!|\phi|\!)\ P\ (\!|\psi|\!)$ can be proved in the partial-correctness calculus we develop in this chapter, we say that the sequent $\vdash_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ is valid.

2. Similarly, if it can be proved in the total-correctness calculus to be developed in this chapter, we say that the sequent $\vdash_{\mathsf{tot}} (\!|\phi|\!)\ P\ (\!|\psi|\!)$ is valid.

Soundness & Completeness

$\vdash_{\mathsf{par}}$ is sound if

$$\models_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ holds whenever } \vdash_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ is valid}$$

for all ϕ, ψ and P; and, similarly, $\vdash_{\mathsf{tot}}$ is sound if

$$\models_{\mathsf{tot}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ holds whenever } \vdash_{\mathsf{tot}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ is valid}$$

for all ϕ, ψ and P. We say that a calculus is *complete* if it is able to prove everything that is true. Formally, $\vdash_{\mathsf{par}}$ is complete if

$$\vdash_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ is valid whenever } \models_{\mathsf{par}} (\!|\phi|\!)\ P\ (\!|\psi|\!) \text{ holds}$$

for all ϕ, ψ and P; and similarly for $\vdash_{\mathsf{tot}}$ being complete.

1. Another version of the factorial program might have been `Fac2`:

```
y = 1;
while (x != 0) {
   y = y * x;
   x = x - 1;
   }
```

Unlike the previous version, it 'consumes' the input x. Nevertheless, it correctly calculates the factorial of x and stores the value in y; and we would like to express that as a Hoare triple. However, it is not a good idea to write $(\!|x \geq 0|\!)$ `Fac2` $(\!|y = x!|\!)$ because, if the program terminates, then x will be 0 and y will be the factorial of the initial value of x.

We need a way of remembering the initial value of x, to cope with the fact that it is modified by the program. Logical variables achieve just that: in the specification $(\!|x = x_0 \wedge x \geq 0|\!)$ `Fac2` $(\!|y = x_0!|\!)$ the x_0 is a logical variable and we read it as being universally quantified in the precondition. Therefore, this specification reads: for all integers x_0, if x equals x_0, $x \geq 0$ and we run the program such that it terminates, then the resulting state will satisfy y equals $x_0!$. This works since x_0 cannot be modified by `Fac2` as x_0 does not occur in `Fac2`.

Definition 4.10 For a Hoare triple $(\!|\phi|\!)\ P\ (\!|\psi|\!)$, its set of logical variables are those variables that are free in ϕ or ψ; and don't occur in P.

$$\frac{(\!|\phi|\!)\ C_1\ (\!|\eta|\!) \qquad (\!|\eta|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ C_1; C_2\ (\!|\psi|\!)}\ \text{Composition}$$

$$\frac{}{(\!|\psi[E/x]|\!)\ x = E\ (\!|\psi|\!)}\ \text{Assignment}$$

$$\frac{(\!|\phi \wedge B|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi \wedge \neg B|\!)\ C_2\ (\!|\psi|\!)}{(\!|\phi|\!)\ \texttt{if}\ B\ \{C_1\}\ \texttt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-statement}$$

$$\frac{(\!|\psi \wedge B|\!)\ C\ (\!|\psi|\!)}{(\!|\psi|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\psi \wedge \neg B|\!)}\ \text{Partial-while}$$

$$\frac{\vdash_{\mathrm{AR}} \phi' \to \phi \qquad (\!|\phi|\!)\ C\ (\!|\psi|\!) \qquad \vdash_{\mathrm{AR}} \psi \to \psi'}{(\!|\phi'|\!)\ C\ (\!|\psi'|\!)}\ \text{Implied}$$

Figure 4.1. Proof rules for partial correctness of Hoare triples.

Proof rules are applied bottom-up.

loop inv.

$\vdash_{AR}$ e.g. $\forall x\,(x \neq x+1)$

For example, let ψ be $x = 6$ and E be 5. Then (ψ) `x = 5` $(\psi[x/E])$ does *not* hold: given a state in which x equals 6, the execution of `x = 5` results in a state in which x equals 5. But $\psi[x/E]$ is the formula $5 = 6$ which holds in no state.

The right way to understand the **Assignment** rule is to think about what you would have to prove about the initial state in order to prove that ψ holds in the resulting state. Since ψ will – in general – be saying something about the value of x, whatever it says about that value must have been true of E, since in the resulting state the value of x is E. Thus, ψ with E in place of x – which says whatever ψ says about x but applied to E – must be true in the initial state.

1. Suppose P is the program `x = 2`. The following are instances of axiom **Assignment**:
   a $(2 = 2)\, P(x = 2)$
   b $(2 = 4)\, P(x = 4)$

   a If you want to prove $x = 2$ after the assignment `x = 2`, then we must be able to prove that $2 = 2$ before it. Of course, 2 is equal to 2, so proving it shouldn't present a problem.
   b If you wanted to prove that $x = 4$ after the assignment, the only way in which it would work is if $2 = 4$; however, unfortunately it is not. More generally, $(\perp)\, x = E\, (\psi)$ holds for any E and ψ – why?

2. Suppose P is $\mathrm{x} = \mathrm{x} + 1$. By choosing various postconditions, we obtain the following instances of the assignment axiom:

a $(x + 1 = 2)\, P(x = 2)$

$\psi : x = 2$

Assignment : $x = x + 1$ E

$\psi(E/x) : x + 1 = 2$

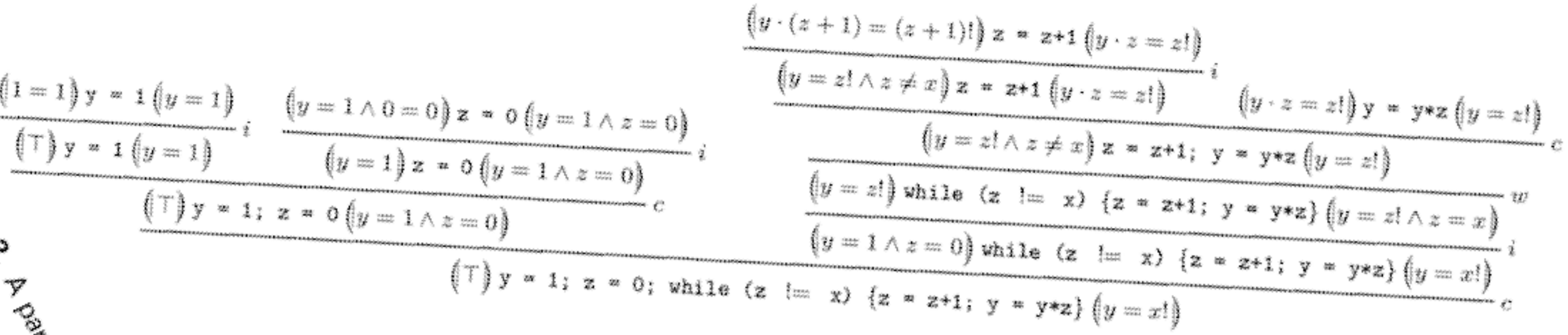


Figure 4.2. A partial-correctness proof for Fac1 in tree form.

a sequence

Program P

$C_1;$
$C_2;$
.
.
.
C_n

$(\!|\phi_0|\!)$ → Pre condition for P

PROOF OUTLINE

$(\!|\phi_0|\!)$

$C_1;$

$(\!|\phi_1|\!)$ justification

$C_2;$

.
.
.

$(\!|\phi_{n-1}|\!)$ justification → mid-condition

$C_n;$

$(\!|\phi_n|\!)$ justification → post-condition for P

Definition 4.12 The process of obtaining ϕ_i from C_{i+1} and ϕ_{i+1} is called computing the weakest precondition of C_{i+1}, given the postcondition ϕ_{i+1}. That is to say, we are looking for the logically weakest formula whose truth at the beginning of the execution of C_{i+1} is enough to guarantee ϕ_{i+1}[4].

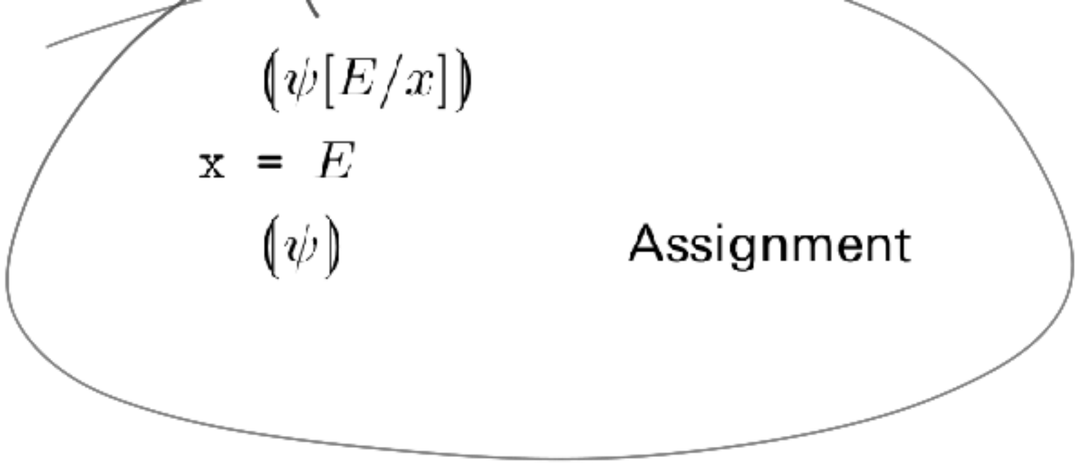
[4] ϕ is weaker than ψ means that ϕ is implied by ψ in predicate logic enlarged with the basic facts about arithmetic: the sequent $\vdash_{AR} \psi \rightarrow \phi$ is valid. We want the weakest formula, because we want to impose as few constraints as possible on the preceding code. In some cases, especially those involving while-statements, it might not be possible to extract the logically weakest formula. We just need one which is sufficiently weak to allow us to complete the proof at hand.

The weakest precondition ϕ' is then checked to see whether it follows from the given precondition ϕ. Thus, we appeal to the Implied rule of Figure 4.1.

$\phi_i \rightarrow \phi'$ is true

$\{\phi'\}\ C_i\ \{\phi_{i+1}\}$

Implied


$(\psi[E/x])$
x = E
(ψ)
Assignment

1. We show that $\vdash_{\mathsf{par}} (\!|y = 5|\!)$ `x = y + 1` $(\!|x = 6|\!)$ is valid:

$$
\begin{array}{ll}
(\!|y = 5|\!) & \\
(\!|y + 1 = 6|\!) & \text{Implied} \\
\texttt{x = y + 1} & \\
(\!|x = 6|\!) & \text{Assignment}
\end{array}
$$

The proof is constructed from the bottom upwards. We start with $(\!|x = 6|\!)$ and, using the assignment axiom, we push it upwards through `x = y + 1`. This means substituting $y + 1$ for all occurrences of x, resulting in $(\!|y + 1 = 6|\!)$. Now, we compare this with the given precondition $(\!|y = 5|\!)$. The given precondition and the arithmetic fact $5 + 1 = 6$ imply it, so we have finished the proof.

Although the proof is constructed bottom-up, its justifications make sense when read top-down: the second line is implied by the first and the fourth follows from the second by the intervening assignment.

$(y = 5) \longrightarrow (y + 1 = 6)$

is true

For the sequential composition of assignment statements

```
z = x;
z = z + y;
u = z;
```

our goal is to show that u stores the sum of x and y after this s
assignments terminates. Let us write P for the code above. Thus, v
prove $\vdash_{\mathsf{par}} (\!|\top|\!)\ P\ (\!|u = x + y|\!)$.

We construct the proof by starting with the postcondition $u =$
pushing it up through the assignments, in reverse order, using the
rule.

– Pushing it up through `u = z` involves replacing all occurrences
resulting in $z = x + y$. We thus have the proof fragment

$(\!|z = x + y|\!)$

```
u = z;
```

$(\!|u = x + y|\!)$ Assignment

– Pushing $z = x + y$ upwards through `z = z + y` involves replacing z by $z + y$,
resulting in $z + y = x + y$.

$(\!|\top|\!)$

$(\!|x + y = x + y|\!)$ Implied

```
z = x;
```

$(\!|z + y = x + y|\!)$ Assignment

```
z = z + y;
```

$(\!|z = x + y|\!)$ Assignment

```
u = z;
```

$(\!|u = x + y|\!)$ Assignment

- Consider the example 'proof'

INCORRECT PROOF

$(x + 1 = x + 1)$

```
x = x + 1;
```

$(x = x + 1)$ Assignment

$x = E$

$E : x + 1$

CORRECT PROOF

$(x + 1 = x + 1 + 1)$

```
x = x + 1;
```

$(x = x + 1)$ Assignment

If-statements. We now consider how to push a postcondition upwards through an if-statement. Suppose we are given a condition ψ and a program fragment `if (`B`) {`C_1`} else {`C_2`}`. We wish to calculate the weakest ϕ such that

$$(\!|\phi|\!)\ \texttt{if (}B\texttt{) \{}C_1\texttt{\} else \{}C_2\texttt{\}}\ (\!|\psi|\!).$$

This ϕ may be calculated as follows.

1. Push ψ upwards through C_1; let's call the result ϕ_1. (Note that, since C_1 may be a sequence of other commands, this will involve appealing to other rules. If C_1 contains another if-statement, then this step will involve a 'recursive call' to the rule for if-statements.)
2. Similarly, push ψ upwards through C_2; call the result ϕ_2.
3. Set ϕ to be $(B \to \phi_1) \wedge (\neg B \to \phi_2)$.

```
a = x + 1;
if (a - 1 == 0)  {
  y = 1;
} else {
  y = a;
}
```

We want to show that $\vdash_{\mathsf{par}} (\!|\top|\!)\ \mathtt{Succ}\ (\!|y = x+1|\!)$ is valid. Note that this program is the sequential composition of an assignment and an if-statement. Thus, we need to obtain a suitable midcondition to put between the if-statement and the assignment.

We push the postcondition $y = x + 1$ upwards through the two branches of the if-statement, obtaining

- ϕ_1 is $1 = x + 1$;
- ϕ_2 is $a = x + 1$;

and obtain the midcondition $(a - 1 = 0 \to 1 = x + 1) \wedge (\neg(a - 1 = 0) \to a = x + 1)$ by appealing to a slightly different version of the rule If-statement:

$$\frac{(\!|\phi_1|\!)\ C_1\ (\!|\psi|\!) \qquad (\!|\phi_2|\!)\ C_2\ (\!|\psi|\!)}{(\!|(B \to \phi_1) \wedge (\neg B \to \phi_2)|\!)\ \mathtt{if}\ B\ \{C_1\}\ \mathtt{else}\ \{C_2\}\ (\!|\psi|\!)}\ \text{If-Statement} \quad (4.7)$$

$\langle\!| \top |\!\rangle$

$\langle\!| ? |\!\rangle$?

```
a = x + 1;
```

$\langle\!| (a - 1 = 0 \to 1 = x + 1) \wedge (\neg(a - 1 = 0) \to a = x + 1) |\!\rangle$?

```
if (a - 1 == 0) {
```

$\langle\!| 1 = x + 1 |\!\rangle$ If-Statement

```
    y = 1;
```

$\langle\!| y = x + 1 |\!\rangle$ Assignment

```
} else {
```

$\langle\!| a = x + 1 |\!\rangle$ If-Statement

```
    y = a;
```

$\langle\!| y = x + 1 |\!\rangle$ Assignment

```
}
```

$\langle\!| y = x + 1 |\!\rangle$ If-Statement

Continuing this example, we push the long formula above the if-statement through the assignment, to obtain

$$(x + 1 - 1 = 0 \to 1 = x + 1) \wedge (\neg(x + 1 - 1 = 0) \to x + 1 = x + 1) \quad (4.8)$$

$$(x = 0 \to 1 = x + 1) \wedge (\neg(x = 0) \to x + 1 = x + 1)$$

and both these conjuncts, and therefore their conjunction, are clearly valid implications. The above proof now is completed as:

$(\!|\top|\!)$

$(\!|(x + 1 - 1 = 0 \to 1 = x + 1) \wedge (\neg(x + 1 - 1 = 0) \to x + 1 = x + 1)|\!)$ Implied

```
a = x + 1;
```

$(\!|(a - 1 = 0 \to 1 = x + 1) \wedge (\neg(a - 1 = 0) \to a = x + 1)|\!)$ Assignment

```
if (a - 1 == 0) {
```

$(\!|1 = x + 1|\!)$ If-Statement

```
    y = 1;
```

$(\!|y = x + 1|\!)$ Assignment

```
} else {
```

$(\!|a = x + 1|\!)$ If-Statement

```
    y = a;
```

$(\!|y = x + 1|\!)$ Assignment

```
}
```

$(\!|y = x + 1|\!)$ If-Statement

While-statements. Recall that the proof rule for partial correctness of while-statements was presented in the following form in Figure 4.1 – here we have written η instead of ψ:

$$\frac{(\!|\eta \wedge B|\!)\ C\ (\!|\eta|\!)}{(\!|\eta|\!)\ \texttt{while}\ B\ \{C\}\ (\!|\eta \wedge \neg B|\!)}\ \textsf{Partial-while.} \tag{4.9}$$

$$(\!|\phi|\!)\ \texttt{while}\ (B)\ \{\texttt{C}\}\ (\!|\psi|\!) \tag{4.10}$$

for some ϕ and ψ which are not related in that way. How can we use Partial-while in a situation like this?

The answer is that we must *discover* a suitable η, such that

loop-invariant

1. $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$,
2. $\vdash_{\mathrm{AR}} \eta \wedge \neg B \rightarrow \psi$ and
3. $\vdash_{\mathsf{par}} (\!|\eta|\!)\ \texttt{while}\ (B)\ \{C\}\ (\!|\eta \wedge \neg B|\!)$

Definition 4.15 An invariant of the while-statement `while (B) {C}` is a formula η such that $\models_{\mathsf{par}} (\!| \eta \wedge B |\!)\ C\ (\!| \eta |\!)$ holds; i.e. for all states l, if η and B are true in l and C is executed from state l and terminates, then η is again true in the resulting state.

Example 4.16 Consider the program `Fac1` from page 262, annotated with location labels for our discussion:

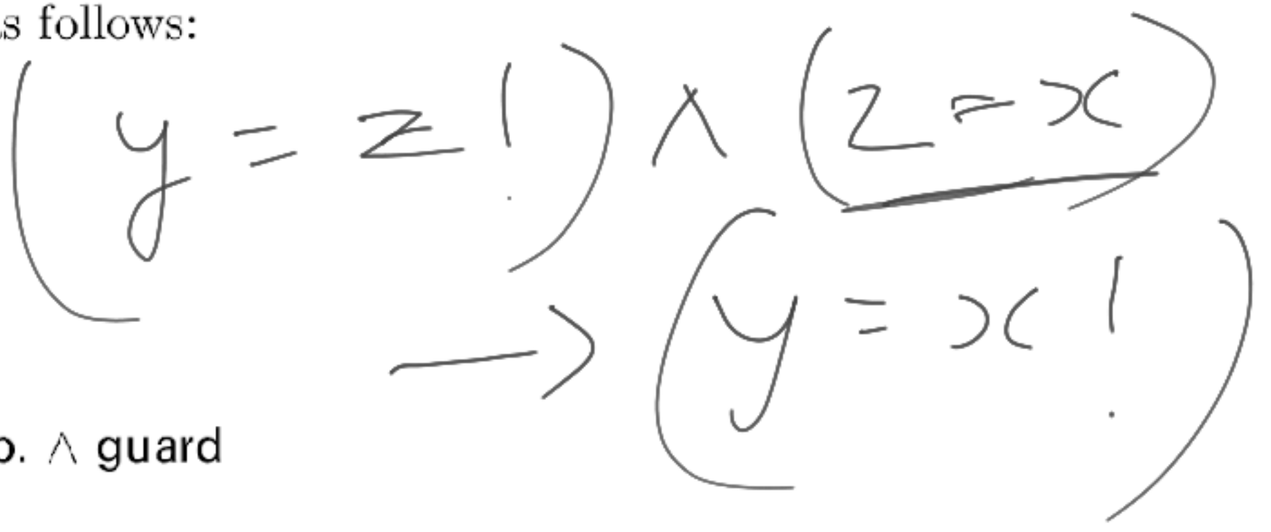
```
    y = 1;
    z = 0;
l1: while (z != x) {
      z = z + 1;
      y = y * z;
l2: }
```

after iteration	z at `l1`	y at `l1`	B at `l1`
0	0	1	true
1	1	1	true
2	2	2	true
3	3	6	true
4	4	24	true
5	5	120	true
6	6	720	false

How should we use the while-rule in proof tableaux? We need to think about how to push an arbitrary postcondition ψ upwards through a while-statement to meet the precondition ϕ. The steps are:

1. Guess a formula η which you hope is a suitable invariant.
2. Try to prove that $\vdash_{\mathrm{AR}} \eta \wedge \neg B \rightarrow \psi$ and $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$ are valid, where B is the boolean guard of the while-statement. If both proofs succeed, go to 3. Otherwise (if at least one proof fails), go back to 1.
3. Push η upwards through the body C of the while-statement; this involves applying other rules dictated by the form of C. Let us name the formula that emerges η'.
4. Try to prove that $\vdash_{\mathrm{AR}} \eta \wedge B \rightarrow \eta'$ is valid; this proves that η is indeed an invariant. If you succeed, go to 5. Otherwise, go back to 1.
5. Now write η above the while-statement and write ϕ above that η, annotating that η with an instance of Implied based on the successful proof of the validity of $\vdash_{\mathrm{AR}} \phi \rightarrow \eta$ in 2. Mission accomplished!

Example 4.17 We continue the example of the factorial. The partial proof obtained by pushing $y = x!$ upwards through the while-statement – thus checking the hypothesis that $y = z!$ is an invariant – is as follows:



```
y = 1;
z = 0;
```

$\{\!|y = z!|\!\}$?

```
while (z != x) {
```

$\{\!|y = z! \wedge z \neq x|\!\}$ Invariant Hyp. $\wedge$ guard

$\{\!|y \cdot (z+1) = (z+1)!|\!\}$ Implied

```
    z = z + 1;
```

$\{\!|y \cdot z = z!|\!\}$ Assignment

```
    y = y * z;
```

$\{\!|y = z!|\!\}$ Assignn

```
}
```

$\{\!|y = x!|\!\}$?

Whether $y = z!$ is a suitable invariant depends on three things:

- The ability to prove that it is indeed an invariant, i.e. that $y = z!$ implies $y \cdot (z + 1) = (z + 1)!$. This is the case, since we just multiply each side of $y = z!$ by $z + 1$ and appeal to the inductive definition of $(z + 1)!$ in Example 4.2.
- The ability to prove that η is strong enough that it and the negation of the boolean guard together imply the postcondition; this is also the case, for $y = z!$ and $x = z$ imply $y = x!$.
- The ability to prove that η is weak enough to be established by the code leading up to the while-statement. This is what we prove by continuing to push the result upwards through the code preceding the while-statement.

$(\!|\top|\!)$

$(\!|1 = 0!|\!)$ Implied

```
y = 1;
```

$(\!|y = 0!|\!)$ Assignment

```
z = 0;
```

$(\!|y = z!|\!)$ Assignment

```
while (z != x) {
```

$(\!|y = z! \wedge z \neq x|\!)$ Invariant Hyp. $\wedge$ guard

$(\!|y \cdot (z + 1) = (z + 1)!|\!)$ Implied

```
    z = z + 1;
```

$(\!|y \cdot z = z!|\!)$ Assignment

```
    y = y * z;
```

$(\!|y = z!|\!)$ Assignment

```
}
```

$(\!|y = z! \wedge \neg(z \neq x)|\!)$ Partial-while

$(\!|y = x!|\!)$ Implied