What are we going to do?

I Languages and decision problems.

I Automated theorem proving.

I Revision: the language of all propositional formulas; the first order language *L*(*C,F,R*).

I Revision: the language of all propositional tautologies. Truth Tables.

I First order structures. Model Checking *S* |= *φ*.

I Validities of *L*(*C,F,R*) and theorem proving.

I Axiomatic Hilbert systems (propositional and predicate)

I tableau (propositional and predicate)

I Main results

I Soundness

I Completeness

I Compactness

I G¨odel’s incompleteness theorem for arithmetic

I Applications

I Modal Logic

I Temporal Logic

I Propositional Dynamic Logic

2

# Reading

I “Logic: an introduction to elementary logic” by W Hodges, Penguin, 1977.

I Logic and Discrete Mathematics: A Concise Introduction by

Willem Conradie and Valentin Goranko, and published by John Wiley, 2015) I “A friendly introduction to mathematical logic” by C Leary, Prentice-Hall, 2000. I “Logic for Computer Scientists” by S Reeves and M Clarke, Addison-Wesley, 1999.

I See lecture notes for more reading.

What is Science?

What is Science?

I The project to change lead into gold

I The project to change lead into gold

I The Newtonian theory of planetary motion I The project to change lead into gold

I The Newtonian theory of planetary motion I Theory of musical harmony

I The project to change lead into gold

I The Newtonian theory of planetary motion I Theory of musical harmony

I Homeopathy

I The project to change lead into gold

I The Newtonian theory of planetary motion I Theory of musical harmony I Homeopathy

I Theories of history

I The project to change lead into gold

I The Newtonian theory of planetary motion I Theory of musical harmony I Homeopathy

I Theories of history

I The theory of climate change

I The project to change lead into gold

I The Newtonian theory of planetary motion I Theory of musical harmony I Homeopathy

I Theories of history

I The theory of climate change I Economic planning?

What is Science?

What is Science?

Evidence + Logic.



Formalise mathematics.

1. Find a set of axioms for mathematics and show that the axioms do not contradict themselves (consistency)



Formalise mathematics.

1. Find a set of axioms for mathematics and show that the axioms do not contradict themselves (consistency)
2. Show that these axioms are complete, i.e. they prove all the true statements of mathematics



Formalise mathematics.

1. Find a set of axioms for mathematics and show that the axioms do not contradict themselves (consistency)
2. Show that these axioms are complete, i.e. they prove all the true statements of mathematics
3. Find an algorithm that determines whether a formula is true or not.

# Logic in Computer Science

I For programme specification, e.g. Z.

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG.

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG.

I for programme verification

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG. I for programme verification

I for programme design

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG. I for programme verification I for programme design

I very important in Artificial Intelligence

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG. I for programme verification I for programme design

I very important in Artificial Intelligence

I knowledge representation

I For programme specification, e.g. Z.

I for programming languages, e.g. PROLOG. I for programme verification I for programme design

I very important in Artificial Intelligence

I knowledge representation

I databases (SQL)

# Formal Logic

Three parts.

I Syntax: A language. Grammar. How you write things down!

I Semantics: Interpreting the language. What you have written down means!

I Inference or proof system: Deduction. Proofs. A syntactic device for proving true statements. How to reason with what you have!

# Propositional Logic

*prop* ::= *p*|*q*|*r*|*...*

*fm* ::= *prop*|¬*fm*|(*fm* ◦ *fm*)*,* where ◦ is ∧*,*∨ or →

In other words, using negation, conjunction and disjunction we build formulas from propositions such as *p*, *q*, *r*, etc.

Literal: A proposition or its negation (e.g. *p*, *q*, ¬*r*, but not *p* ∧ *r*)

# Main Connective

The connective with the largest scope (i.e. one that is not in the scope of any other connective), the connective we would start our evaluation with

((*p* ∧ *q*) ∨ ¬(*q* → *r*))

# Propositional Parser

Given a string ((*p* ∧ *q*) ∨ ¬(*q* → *r*)) your parser has to work out that the main connective (∨) is the character with index 7, and that the two parts are (*p* ∧ *q*) and ¬(*q* → *r*).

# Propositional Logic. Semantics

A valuation *v* maps propositions to {>*,*⊥}. *v* extends to a unique truth-function (also called *v*) satisfying

|  |  |  |
| --- | --- | --- |
| *v*(¬*φ*) = > | ⇐⇒ | *v*(*φ*) = ⊥ |
| *v*(*φ* ∧ *ψ*) = > | ⇐⇒ | *v*(*φ*) = *v*(*ψ*) = > |
| *v*(*φ* ∨ *ψ*) = > | ⇐⇒ | *v*(*φ*) = > or *v*(*ψ*) = > |
| *v*(*φ* → *ψ*) = > | ⇐⇒ | *v*(*ψ*) = > or *v*(*φ*) = ⊥ |

# Validity, Satisfiability, Equivalence

We say that:

I *φ* is valid if *v*(*φ*) = > for all possible valuations *v* (i.e. it is always true).

I *φ* is satisfiable if *v*(*φ*) = > for at least one valuation *v* (i.e. it is true at least once).

I *φ* and *ψ* are logically equivalent, written down as *φ* ≡ *ψ*, if and only if for every *v*, *v*(*φ*) = *v*(*ψ*)

Every valid formula is satisfiable, but not necessarily the other way around!

# Predicate Logic. Syntax, Revision

Language *L*(*C,F,P*).

I *C* is a set of constant symbols

I *F* is a set of function symbols. We may write *f n* to indicate that *f* is an n-ary function symbol.

I *P* is a non-empty set of predicate symbols. We may write *pn* to indicate an n-ary predicate symbol.

|  |  |  |
| --- | --- | --- |
| *tm* | ::= | v|c : v ∈ Var*,* c ∈ C|fn(tm*,*tm*,...,*tm) : fn ∈ F |
| atom | ::= | pn(tm0*,*tm1*,...,*tmn−1) : pn ∈ P |
| fm | ::= | atom|¬fm|(fm0 ∨ fm1)|∃vfm : v ∈ Var |

Var is a set of variable symbols.

# Predicate Logic

Signature L(C*,*F*,*P), where C is a set of constants, F is a set of function symbols and P is a set of predicate symbols. Each function and predicate symbol has an arity. Var is a countable set of variables.

I Term

*τ* ::= c (∈ C)|v (∈ Var)|f(*τ*0*,...,τ*n−1) (f ∈ F is n-ary)

e.g. 3 + (x ∗ 2), where 2*,*3 ∈ C*,* +*,*∗ ∈ F (both binary) and x ∈ Var.

# Predicate Formula

I Atomic

## R(τ0,...,τn−1)

Where *τ*i is a term (i *<* n) and R ∈ P is an n-ary predicate.

E.g. x + y *<* 2 ∗ y − 1, where *<* is a binary predicate (written infix).

I Formula *φ* ::= Atom|¬*φ*|(*φ* ∨ *φ*0)|∃x*φ*

Write ∀x*φ* as an abbreviation of ¬∃x¬*φ*. e.g. (¬∃xR(x) → ∀y(S(x*,*y) ∨ S(y*,*x))).

# *L*-structure

(D*,*I) where D is any non-empty set (the domain) and I interprets constants, functions and predicates

I I : C → D

I I(f) : Dn → D (if f ∈ F is n-ary) I I(R) ⊆ Dn (if R ∈ P is n-ary).

Variable Assignment

A : Var → D

# Evaluating Terms

Let S = (*D,I*) be a structure, *A* : *Var* → *D* be a variable assignment.

[*c*]S*,A* = *I*(*c*)

[*x*]S*,A* = *A*(*x*)

[*f* (*τ*0*,...,τn*−1)]S*,A* = *I*(*f* )([*τ*0]S*,A,...,*[*τn*−1]S*,A*)

# Formulas

S*,A* |= *R*(*τ*0*,...,τn*−1) ⇐⇒ ([*τ*0]S*,A,...,*[*τn*−1]S*,A*) ∈ *I*(*R*)

S*,A* |= ¬*φ* ⇐⇒ S*,A* 6|= *φ*

S*,A* |= (*φ* ∨ *φ*0) ⇐⇒ S*,A* |= *φ* or S*,A* |= *φ*0

S*,A* |= ∃*xφ* ⇐⇒ S*,A*[*x* 7→ *d*] |= *φ* for some *d* ∈ *D*

# Validity

Let S = (*D,I*) be an *L*-structure, *φ* a formula.

|  |  |  |
| --- | --- | --- |
| Statement | Def. | Written |
| *φ* is valid in S | for all *A* : *Var* → *D* we have S*,A* |= *φ* | S |= *φ* |
| *φ* is valid | for all *L*-structures S we have S |= *φ* | |= *φ* |

Similarly, *φ* is satisfiable in S if there is some *A* : *Var* → *D* such that S*,A* |= *φ*, and *φ* is simply satisfiable if it is satisfiable in some structure. *φ* is not valid if and only if ¬*φ* is satisfiable.

Propositional Proof Systems

Proof system: A system for determining the validity of formulas.

# Propositional Proof Systems

Proof system: A system for determining the validity of formulas.

Obvious system: Write down the truth table for *φ* and check that all rows give value >.

Propositional Proof Systems

Problem: Takes a long time (exponential time)

# Propositional Proof Systems

Problem: Takes a long time (exponential time) *φ* : *p*1 ∨ *p*2 ∨ *...* ∨ *p*49 ∨ *p*50

Problem: Takes a long time (exponential time) *φ* : *p*1 ∨ *p*2 ∨ *...* ∨ *p*49 ∨ *p*50

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| *p*1 | *p*2 | *...* | *p*49 | *p*50 | *φ* |
| > | > | *...* | > | > | > |
| > | > | *... ...* | > | ⊥ | > |

Problem: Takes a long time (exponential time) *φ* : *p*1 ∨ *p*2 ∨ *...* ∨ *p*49 ∨ *p*50

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  |  |  | | --- | --- | --- | --- | --- | | *p*1 | *p*2 | *...* | *p*49 *p*50 | *φ* | | > | > | *...* | > > | > | | > | > | *... ...* | > ⊥ | > | | ⊥ | ⊥ | *...* | ⊥ ⊥ | ⊥ | |  |

Only one out of 1,125,899,906,842,624 (more than a quadrillion!) valuations makes *φ* false!

Proof system: A system for determining the validity of formulas.

~~Obvious system: Write down the truth table for~~ *~~φ~~* ~~and check that all rows give value >.~~

Proof system: A system for determining the validity of formulas.

~~Obvious system: Write down the truth table for~~ *~~φ~~* ~~and check that all rows give value >.~~

Another way: Manipulate and analyze the syntax of the formula in order to find out whether there is something that can falsify it.

Another way: Manipulate and analyze the syntax of the formula in order to find out whether there is something that can falsify it.

Another way: Manipulate and analyze the syntax of the formula in order to find out whether there is something that can falsify it.

Problem: How to ensure that the syntactical changes lead to results that make sense on the semantical level?

Another way: Manipulate and analyze the syntax of the formula in order to find out whether there is something that can falsify it.

Problem: How to ensure that the syntactical changes lead to results that make sense on the semantical level?

Solution: Make sure that the system we are using is sound and complete.

# Propositional Proof Systems

|= *φ* means *φ* is valid

` *φ* means there is a proof of *φ*

# Propositional Proof Systems

|= *φ* means *φ* is valid

` *φ* means there is a proof of *φ*

I Soundness: the system can prove only valid things:

` *φ* ⇒ |= *φ*

# Propositional Proof Systems

|= *φ* means *φ* is valid

` *φ* means there is a proof of *φ*

I Soundness: the system can prove only valid things:

## ` φ ⇒ |= φ

I Completeness: if something is valid that the system can prove it:

|= *φ* ⇒ ` *φ*

# Axiomatic Proof Systems

Fix a propositional language with only → and ¬, and no double negations ¬¬.

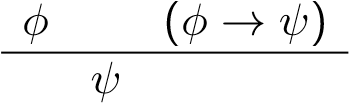
Axiom Schema

1. (*p* → (*q* → *p*))
2. ((*p* → (*q* → *r*)) → ((*p* → *q*) → (*p* → *r*)))
3. ((¬*p* → ¬*q*) → (*q* → *p*)).

An axiom may be obtained by substitution any formulas in place of *p,q,r* above.

# Inference Rule

Modus Ponens.



If you have proved *φ* and you have proved (*φ* → *ψ*) then you may deduce *ψ*.

# Proofs

A proof is a sequence of formulas

## φ0,φ1,φ2,...,φn

such that for *i* ≤ *n*, *φi* is either an instance of an axiom or it is obtained by modus ponens from *φj,φk* (some *j,k < i*), e.g., if *φk* = (*φj* → *φi*).

If such a proof exists, *φn* is called a theorem and we may write

` *φn*

# Proof Example. ` (*p*→*p*)

1. ((*p* → ((*p* → *p*) → *p*)) → ((*p* → (*p* → *p*)) → (*p* → *p*)))

(Ax. II, replace *p,q,r* by *p,*(*p* → *p*) and *p*).

1. (*p* → ((*p* → *p*) → *p*)) (Ax. I, replace *p,q* by *p,*(*p* → *p*)).
2. ((*p* → (*p* → *p*)) → (*p* → *p*)) (M.P., 1, 2).
3. (*p* → (*p* → *p*)) (Ax. I).
4. (*p* → *p*) (M.P., 3, 4).

# Proofs with other connectives

To include double negations, add axioms

1. (*p* → ¬¬*p*) and (¬¬*p* → *p*).

For disjunction and conjunctions, add axioms

1. ((*p* ∨ *q*) ↔ (¬*p* → *q*)),
2. ((*p* ∧ *q*) ↔ ¬(*p* → ¬*q*)).

# Proofs with assumptions

Write

## Γ ` φ

if there is a proof of *φ* using assumptions from Γ. I.e. there is a sequence

## φ0,φ1,...,φn

where *φ* = *φn* and for each *i* ≤ *n*, *φi* is either

I an instance of an axiom schema,

I an assumption, *φi* ∈ Γ, or

I obtained from *φj,φk* (some *j,k < i*) by modus ponens.

# Proof example 2: {⊥}`*p*

or, rather {¬(*q* → *q*)} ` *p*.

1. (*q* → *q*) (previous example)
2. ((*q* → *q*) → ¬¬(*q* → *q*)) (Ax. IV)
3. ¬¬(*q* → *q*) (M.P. 5,6)
4. (¬¬(*q* → *q*) → (¬*p* → ¬¬(*q* → *q*))) (Ax. I)
5. (¬*p* → ¬¬(*q* → *q*)) (M.P. 7,8)
6. (¬*p* → ¬¬(*q* → *q*)) → (¬(*q* → *q*) → *p*) (Ax. III)
7. (¬(*q* → *q*) → *p*) (M.P. 9, 10) 12 ¬(*q* → *q*) (assumption) 13 *p* (M.P. 11,12).

# Soundness and Completeness

Soundness Check all axioms are valid. Check that if *φ* is valid and (*φ* → *ψ*) is valid then *ψ* is valid. Deduce (by induction on length of proof) that all provable formulas are valid,

` *φ* ⇒ |= *φ*

# Soundness and Completeness

Soundness Check all axioms are valid. Check that if *φ* is valid and (*φ* → *ψ*) is valid then *ψ* is valid. Deduce (by induction on length of proof) that all provable formulas are valid,

` *φ* ⇒ |= *φ*

Completeness |= *φ* ⇒ ` *φ*

A bit harder.

# Soundness and Completeness

Soundness Check all axioms are valid. Check that if *φ* is valid and (*φ* → *ψ*) is valid then *ψ* is valid. Deduce (by induction on length of proof) that all provable formulas are valid,

` *φ* ⇒ |= *φ*

Completeness |= *φ* ⇒ ` *φ*

A bit harder.

Termination Not necessarily.

# Propositional Tableaux

I Given the formula *φ*, a tableau can tell us whether it is satisfiable or not

I Given the formula *φ*, a tableau can tell us whether it is satisfiable or not

I It does so by decomposing the formulae according to certain rules to the point that only literals are left

I Given the formula *φ*, a tableau can tell us whether it is satisfiable or not

I It does so by decomposing the formulae according to certain rules to the point that only literals are left

I A tableau will be deemed closed or not depending on the nature of the literals we receive

I Given the formula *φ*, a tableau can tell us whether it is satisfiable or not

I It does so by decomposing the formulae according to certain rules to the point that only literals are left

I A tableau will be deemed closed or not depending on the nature of the literals we receive

I If the tableau closes then *φ* is unsatisfiable

I Given the formula *φ*, a tableau can tell us whether it is satisfiable or not

I It does so by decomposing the formulae according to certain rules to the point that only literals are left

I A tableau will be deemed closed or not depending on the nature of the literals we receive

I If the tableau closes then *φ* is unsatisfiable

I If the tableau never closes then *φ* is satisfiable

# The idea

To test whether ((*φ* ∧ ¬*φ*) ∨ (*ψ* ∧ ¬*ψ*)) is satisfiable ((*φ* ∧ ¬*φ*) ∨ (*ψ* ∧ ¬*ψ*)) X

(*φ* ∧ ¬*φ*) X

*φ*

¬*φ*

⊕

(*ψ* ∧ ¬*ψ*) X

*ψ*

¬*ψ*

⊕

# The idea

To test whether ((*φ* ∧ ¬*φ*) ∨ (*ψ* ∧ ¬*ψ*)) is satisfiable

((

*φ*

∧¬

*φ*

)

∨

(

*ψ*

∧¬

*ψ*

))

X

(

*φ*

∧¬

*φ*

)

X

*φ*

¬

*φ*

⊕

(

*ψ*

∧¬

*ψ*

)

X

*ψ*

¬

*ψ*

⊕

Both branches close ⇒ not satisfiable.

# Tableaux

I Formally, a tableau *T* is a type of binary tree where every node is labelled by a formula.

# Tableaux

I Formally, a tableau *T* is a type of binary tree where every node is labelled by a formula.

I At first, *T* has only one node, containing the formula *φ* whose satisfiability we are investigating.

# Tableaux

I Formally, a tableau *T* is a type of binary tree where every node is labelled by a formula.

I At first, *T* has only one node, containing the formula *φ* whose satisfiability we are investigating.

I Every formula in the tableau, except literals, gets “expanded” and ticked.

I If a branch of *T* contains *p* and ¬*p* then it is closed. Otherwise it is open.

I If every branch of *T* is closed then *T* is closed. Otherwise it is open.

# Tableau Expansion Rules

*α* formulas

(*φ* ∧ *ψ*)

*ψ*

*ψ*

We observe that (*φ* ∧ *ψ*) is true if and only if *φ* is true and *ψ* is true.

*α* formulas

(*φ* ∧ *ψ*) X

*ψ*

*ψ*

We observe that (*φ* ∧ *ψ*) is true if and only if *φ* is true and *ψ* is true.

Means that nodes corresponding to *φ* and *ψ* are added at every leaf of *T* below the current node (*φ* ∧ *ψ*).

# Tableau Expansion Rules

*α* formulas

Three other kinds of *α* formulas.

¬¬*φ*

*φ*

*α* formulas

Three other kinds of *α* formulas.

¬¬*φ* ¬(*φ* ∨ *ψ*) X

*φ* ¬*φ*

¬*ψ*

Observe that ¬(*φ* ∨ *ψ*) ≡ ¬*φ* ∧ ¬*ψ*

*α* formulas

Three other kinds of *α* formulas.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| ¬¬*φ*  *φ* | ¬(*φ* ∨ *ψ*)  ¬*φ*  ¬*ψ* | X | ¬(*φ* → *ψ*)  *φ*  ¬*ψ* | X |

Observe that ¬(*φ* ∨ *ψ*) ≡ ¬*φ* ∧ ¬*ψ*

Observe that ¬(*φ* → *ψ*) ≡ ¬(¬*φ* ∨ *ψ*) ≡ *φ* ∧ ¬*ψ*

*β* formulas

(*φ* ∨ *ψ*)

*φ ψ*

We observe that (*φ* ∨ *ψ*) is true if and only if *φ* is true or *ψ* is true, but not necessarily both of them.

*β* formulas

(*φ* ∨ *ψ*) X

*φ ψ*

We observe that (*φ* ∨ *ψ*) is true if and only if *φ* is true or *ψ* is true, but not necessarily both of them.

The first one means that two branches (one labelled *φ* the other labelled *ψ*) are added at every leaf below the current node (*φ* ∨ *ψ*).

*β* formulas

There are two other kinds of *β* formulas.

¬(*φ* ∧ *ψ*) X

¬*φ* ¬*ψ*

Observe that ¬(*φ* ∧ *ψ*) ≡ ¬*φ* ∨ ¬*ψ*

*β* formulas

There are two other kinds of *β* formulas.

¬(*φ* ∧ *ψ*) X (*φ* → *ψ*) X

¬*φ* ¬*ψ* ¬*φ ψ*

Observe that ¬(*φ* ∧ *ψ*) ≡ ¬*φ* ∨ ¬*ψ*

Observe that (*φ* → *ψ*) ≡ ¬*φ* ∨ *ψ*

I (*p* ∨ *r*) I ¬*q*

I ¬((*p* ∨ *q*) → *r*)

I ((*p* → *r*) ∨ (*q* → *r*))

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*))

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q*

I ¬((*p* ∨ *q*) → *r*)

I ((*p* → *r*) ∨ (*q* → *r*))

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*))

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) I ((*p* → *r*) ∨ (*q* → *r*))

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*))

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*))

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*))

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*)) (*β* rule)

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*))

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*)) (*β* rule)

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*)) (*α* rule)

I ¬(*p* → *r*)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*)) (*β* rule)

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*)) (*α* rule)

I ¬(*p* → *r*) (*α* rule)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*)) (*β* rule)

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*)) (*α* rule) I ¬(*p* → *r*) (*α* rule)

(*β* rule)

I (*p* ∨ *r*) (*β* rule)

I ¬*q* (none - literal)

I ¬((*p* ∨ *q*) → *r*) (*α* rule)

I ((*p* → *r*) ∨ (*q* → *r*)) (*β* rule)

I ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨ *q*) → *r*)) (*α* rule) I ¬(*p* → *r*) (*α* rule)

(*β* rule)

(none - literal)

I ¬(*p* ∧ *q*)

I ¬((*p* → *r*) ∨ (*q* → *r*))

I (*p* ∨ *q*)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*)))

I *p*

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*))

I (*p* ∨ *q*)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*)))

I *p*

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*)))

I *p*

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*)))

I *p*

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*))) (*α* rule)

I *p*

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*))) (*α* rule)

I *p* (none - literal)

I (*p* → *r*)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*))) (*α* rule)

I *p* (none - literal)

I (*p* → *r*) (*β* rule)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*))) (*α* rule)

I *p* (none - literal)

I (*p* → *r*) (*β* rule)

(*α* rule)

I ¬(*p* ∧ *q*) (*β* rule)

I ¬((*p* → *r*) ∨ (*q* → *r*)) (*α* rule)

I (*p* ∨ *q*) (*β* rule)

I ¬(((*p* ∧ *q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*))) (*α* rule)

I *p* (none - literal)

I (*p* → *r*) (*β* rule)

(*α* rule)

(*α* rule)

# Tableaux

I If *φ* is at the root of a closed tableau, then *φ* is not satisfiable.

I If *φ* is at the root of a closed tableau, then *φ* is not satisfiable.

I Being at the root of an open tableau is not enough to be considered satisfiable!

I If *φ* is at the root of a closed tableau, then *φ* is not satisfiable.

I Being at the root of an open tableau is not enough to be considered satisfiable!

## ((φ ∧ ¬φ) ∨ (ψ ∧ ¬ψ)) X

## (φ ∧ ¬φ) (ψ ∧ ¬ψ)

This tableau is still open, but the formula ((*φ* ∧ ¬*φ*) ∨ (*ψ* ∧ ¬*ψ*)) clearly cannot be considered satisfiable!

I A tableau is complete if every node is either ticked (already expanded) or a literal.

I If *φ* is at the root of a complete open tableau, then *φ* is satisfiable.

# Propositional Tableaux

Remember: In a proof system we are interested in validity, not just satisfiability. Fortunately, we know that:

*φ* is satisfiable ⇐⇒ ¬*φ* is not valid *φ* is valid ⇐⇒ ¬*φ* is not satisfiable

# Propositional Tableaux

Remember: In a proof system we are interested in validity, not just satisfiability. Fortunately, we know that:

*φ* is satisfiable ⇐⇒ ¬*φ* is not valid *φ* is valid ⇐⇒ ¬*φ* is not satisfiable

Solution: In order to test whether a formula *φ* is valid we see whether the tableau for ¬*φ* closes

1. ¬(((*p* ∧*q*) → *r*) → ((*p* → *r*) ∨ (*q* → *r*)))

*α*(1)

1. ((*p* ∧*q*) → *r*)
2. ¬((*p* → *r*) ∨ (*q* → *r*))

(10) ¬*p*

*α*(6)

1. *p*
2. ¬*r*

⊕

(4) ¬(*p* ∧*q*) *β*(2)

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*β*(4) (11) ¬*q*

*α*(6)

(14) *p* (15) ¬*r*

*α*(7)

(5) *r*

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*α*(8)

* 1. *p*
  2. ¬*r*

⊕

* 1. ((*p* ∧*q*) → *r*)
  2. ¬((*p* → *r*) ∨ (*q* → *r*))

1. ¬*p*

*α*(6)

1. *p*
2. ¬*r*

⊕

(4) ¬(*p* ∧*q*) *β*(2)

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*β*(4) (11) ¬*q*

*α*(6)

(14) *p* (15) ¬*r*

*α*(7)

(5) *r*

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*α*(8)

* 1. *p*
  2. ¬*r*

⊕

|  |
| --- |
| (2) ((*p* ∧*q*) → *r*) X  (3)  ¬  ((  *p*  →  *r*  )  ∨  (  *q*  →  *r*  )) |

1. ¬*p*

*α*(6)

1. *p*
2. ¬*r*

⊕

(4) ¬(*p* ∧*q*) *β*(2)

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*β*(4) (11) ¬*q*

*α*(6)

(14) *p* (15) ¬*r*

*α*(7)

(5) *r*

*α*(3)

1. ¬(*p* → *r*)
2. ¬(*q* → *r*)

*α*(8)

* 1. *p*
  2. ¬*r*

⊕

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(3)

¬

((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(4)

¬

(

*p*

∧

*q*

)

(6)

¬

(

*p*

→

*r*

)

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

*α*

(3)

*α*

(3)

|  |  |  |  |
| --- | --- | --- | --- |
| (10) ¬*p*  *α*(6)   1. *p* 2. ¬*r*   ⊕ | (7) ¬(*q* → *r*)  *β*(4) | (11) ¬*q*  *α*(6)   1. *p* 2. ¬*r*   *α*(7) | (9) ¬(*q* → *r*)  *α*(8)   1. *p* 2. ¬*r*   ⊕ |

*α*(3)

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(3)

¬

((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(4)

¬

(

*p*

∧

*q*

)

X

(6)

¬

(

*p*

→

*r*

)

(7)

¬

(

*q*

→

*r*

)

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

(9)

¬

(

*q*

→

*r*

)

*α*

(3)

*α*

(8)

|  |  |  |  |
| --- | --- | --- | --- |
| (10) ¬*p*  *α*(6)   1. *p* 2. ¬*r*   ⊕ | *β*(4) | (11) ¬*q*  *α*(6)   1. *p* 2. ¬*r*   *α*(7) | 1. *p* 2. ¬*r*   ⊕ |

*α*(3)

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(3)

¬

((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(4)

¬

(

*p*

∧

*q*

)

X

(6)

¬

(

*p*

→

*r*

)

X

(7)

¬

(

*q*

→

*r*

)

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

(9)

¬

(

*q*

→

*r*

)

*α*

(3)

*α*

(8)

|  |  |  |  |
| --- | --- | --- | --- |
| (10) ¬*p*  *α*(6)   1. *p* 2. ¬*r*   ⊕ | *β*(4) | (11) ¬*q*  *α*(6)   1. *p* 2. ¬*r*   *α*(7) | 1. *p* 2. ¬*r*   ⊕ |

*α*(3)

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(3)

¬

((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(4)

¬

(

*p*

∧

*q*

)

X

(6)

¬

(

*p*

→

*r*

)

X

(7)

¬

(

*q*

→

*r*

)

X

(10)

¬

*p*

(12)

*p*

(13)

¬

*r*

⊕

*β*

(4)

(11)

¬

*q*

(14)

*p*

(15)

¬

*r*

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

(9)

¬

(

*q*

→

*r*

)

(18)

*p*

(19)

¬

*r*

⊕

*α*

(3)

*α*

(6)

*α*

(6)

*α*

(7)

*α*(8)

(3) ¬((*p* → *r*) ∨ (*q* → *r*)) X

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(4)

¬

(

*p*

∧

*q*

)

X

(6)

¬

(

*p*

→

*r*

)

X

(7)

¬

(

*q*

→

*r*

)

X

(10)

¬

*p*

(12)

*p*

(13)

¬

*r*

⊕

*β*

(4)

(11)

¬

*q*

(14)

*p*

(15)

¬

*r*

(16)

*q*

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

X

(9)

¬

(

*q*

→

*r*

)

(18)

*p*

(19)

¬

*r*

⊕

*α*

(3)

*α*

(6)

*α*

(6)

*α*

(7)

*α*

(3)

*α*

(8)

(17) ¬*r*

⊕

(3) ¬((*p* → *r*) ∨ (*q* → *r*)) X

Closed tableau —

(2) ((

*p*

∧

*q*

)

→

*r*

)

X

(4)

¬

(

*p*

∧

*q*

)

X

(6)

¬

(

*p*

→

*r*

)

X

(7)

¬

(

*q*

→

*r*

)

X

(10)

¬

*p*

(12)

*p*

(13)

¬

*r*

⊕

*β*

(4)

(11)

¬

*q*

(14)

*p*

(15)

¬

*r*

(16)

*q*

*β*

(2)

(5)

*r*

(8)

¬

(

*p*

→

*r*

)

X

(9)

¬

(

*q*

→

*r*

)

(18)

*p*

(19)

¬

*r*

⊕

*α*

(3)

*α*

(6)

*α*

(6)

*α*

(7)

*α*

(3)

*α*

(8)

(17) ¬*r*

⊕

1. ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨*q*) *r*))

*α*(1)

1. ((*p* → *r*) ∨ (*q* → *r*))
2. ¬((*p* ∨*q*) → *r*)
3. (*p* → *r*) *β*(2)

*α*(3)

1. (*p* ∨*q*)
2. ¬*r*

(10) *r β*(4) (11) ¬*p* (14) *r*

(5) (*q* → *r*)

*α*(3)

1. (*p* ∨*q*)
2. ¬*r*

*β*(5) (15) ¬*q*

(4) (*p* → *r*)

*α*(3)

* 1. (*p* ∨*q*)
  2. ¬*r*

1. *r β*(4)

*α*(1)

1. ((*p* → *r*)∨ (*q* → *r*))
2. ¬((*p* ∨*q*) → *r*)

*β*(2) (5) (*q* → *r*)

*α*(3)

* 1. (*p* ∨*q*)
  2. ¬*r*

(11) ¬*p* (14) *r β*(5) (15) ¬*q*

(4) (*p* → *r*)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

*α*

(1)

*α*(3)

1. (*p* ∨*q*)
2. ¬*r*
3. *r β*(4)

*β*(2) (5) (*q* → *r*)

*α*(3)

* 1. (*p* ∨*q*)
  2. ¬*r*

1. ¬*p* (14) *r β*(5) (15) ¬*q*

(4) (*p* → *r*)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

*α*

(1)

*α*(3)

* 1. (*p*∨*q*)
  2. ¬*r*

1. *r β*(4)

*β*(2) (5) (*q* → *r*)

*α*(3)

* 1. (*p*∨*q*)
  2. ¬*r*

1. ¬*p* (14) *r β*(5) (15) ¬*q*

*α*(3)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*q*

)

(7)

¬

*r*

(10)

*r*

*β*

(4)

(11)

¬

*p*

*β*

(2)

(5) (

*q*

→

*r*

)

(8) (

*p*

∨

*q*

)

(9)

¬

*r*

(14)

*r*

*β*

(5)

(15)

¬

*q*

*α*

(1)

*α*

(3)

*α*(3)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*q*

)

X

(7)

¬

*r*

(10)

*r*

*β*

(4)

(11)

¬

*p*

*β*

(2)

(5) (

*q*

→

*r*

)

(8) (

*p*

∨

*q*

)

(9)

¬

*r*

(14)

*r*

*β*

(5)

(15)

¬

*q*

*α*

(1)

*α*

(3)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*q*

)

X

(7)

¬

*r*

(10)

*r*

*β*

(4)

(11)

¬

*p*

*β*

(2)

(5) (

*q*

→

*r*

)

X

(8) (

*p*

∨

*q*

)

(9)

¬

*r*

(14)

*r*

*β*

(5)

(15)

¬

*q*

*α*

(1)

*α*

(3)

*α*

(3)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*q*

)

X

(7)

¬

*r*

(10)

*r*

*β*

(4)

(11)

¬

*p*

*β*

(2)

(5) (

*q*

→

*r*

)

X

(8) (

*p*

∨

*q*

)

X

(9)

¬

*r*

(14)

*r*

*β*

(5)

(15)

¬

*q*

*α*

(1)

*α*

(3)

*α*

(3)

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*q*

)

X

(7)

¬

*r*

(10)

*r*

⊕

*β*

(4)

(11)

¬

*p*

(12)

*p*

⊕

*β*

(6)

(13)

*q*

*β*

(2)

(5) (

*q*

→

*r*

)

X

(8) (

*p*

∨

*q*

)

X

(9)

¬

*r*

(14)

*r*

⊕

*β*

(5)

(15)

¬

*q*

(16)

*p*

*β*

(8)

(17)

*q*

*α*

(1)

*α*

(3)

*α*

(3)

Two open branches — root formula is satisfiable, original formula is not valid.

# Disjunctive Normal Form (DNF)

A DNF formula is a disjunctions of clauses where each clause is a conjunction of literals. E.g.

(*p* ∧ ¬*q* ∧ *r*) ∨ (*p* ∧ *q*) ∨ (¬*p* ∧ ¬*q* ∧ ¬*r*))

Also

*p* ∨ ¬*q* ∨ (*p* ∧ *q*)

Every propositional formula has an equivalent formula in DNF. A CNF formula is a conjunction of clauses where each clause is disjunction of literals. E.g.

(*p* ∨ *q*) ∧ (¬*p* ∨ ¬*r*) ∧ *q*

# Converting a formula to DNF

1. By truth table
2. By logical equivalences (De Morgan’s laws, distribution laws etc.)

# Converting a formula to DNF

1. By truth table
2. By logical equivalences (De Morgan’s laws, distribution laws etc.)
3. By tableau

# Using tableau to find DNF equivalent

I Place *φ* at root of new tableau.

I Expand until tableau *T* is completed (no nodes left to expand).

I For each open branch Θ of *T* let

*C*Θ = ^{literals in Θ}

I Then

*φ* ≡ \_ *C*Θ open branches Θ

Example 2: Is formula (((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨*q*) → *r*)) valid?

(1) ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨*q*) → *r*)) X

(2) ((

*p*

→

*r*

)

∨

(

*q*

→

*r*

))

X

(3)

¬

((

*p*

∨

*q*

)

→

*r*

)

X

(4) (

*p*

→

*r*

)

X

(6) (

*p*

∨

*r*

)

X

(7)

¬

*r*

(10)

*r*

⊕

*β*

(4)

(11)

¬

*p*

(12)

*p*

*β*

(6)

(13)

*q*

*β*

(2)

(5) (

*q*

→

*r*

)

X

(8) (

*p*

∨

*q*

)

X

(9)

¬

*r*

(14)

*r*

⊕

*β*

(5)

(15)

¬

*q*

(16)

*p*

*β*

(8)

(17)

*q*

*α*

(1)

*α*

(3)

*α*

(3)

⊕ ⊕

The DNF for ¬(((*p* → *r*) ∨ (*q* → *r*)) → ((*p* ∨*q*) → *r*)) is (¬*p* ∧*q* ∧¬*r*) ∨ (*p* ∧¬*q* ∧¬*r*)

# First Order Tableaus

I A literal is an atom or its negation (i.e. *rn*(*t*1*,...,tn*) or ¬*rn*(*t*1*,...,tn*) where *r* is a predicate and *ti* is a term)

I A closed term is a term that contains no variables (e.g. constants, functions over constants, etc.)

I We use the same kind of tableau construction that we used for propositional logic...

# First Order Tableaus

I A literal is an atom or its negation (i.e. *rn*(*t*1*,...,tn*) or ¬*rn*(*t*1*,...,tn*) where *r* is a predicate and *ti* is a term)

I A closed term is a term that contains no variables (e.g. constants, functions over constants, etc.)

I We use the same kind of tableau construction that we used for propositional logic...

I ...but we need new expansion rules to deal with the quantifiers!

# Expansion Rules

*α* formulas Add both formulas in one branch at each leaf below current node. Tick current node.

(*φ* ∧ *ψ*) X ¬¬*φ* X ¬(*φ* ∨ *ψ*) X ¬(*φ* → *ψ*) X

*φφ* ¬*φφ*

*ψ* ¬*ψ* ¬*ψ*

*β* formulas Make two separate branches (one with each formula) at each leaf below current node. Tick current node.

¬(*φ* ∧ *ψ*) X *φ* ∨ *ψ* X *φ* → *ψ* X

¬*φ* ¬*ψ φ ψ* ¬*φ ψ*

*δ*

Choose new constant *p* (not included in tableau so far). Add formula at each leaf below current node. Tick current node.

## ∃xφ X

## φ(p/x)

*δ*

Choose new constant *p* (not included in tableau so far). Add formula at each leaf below current node. Tick current node.

## ∃xφ X ¬∀xφ X

### φ(p/x) ¬φ(p/x)

Observe: ¬∀*xφ* ≡ ∃*x*¬*φ*

*γ*

Pick any closed term *t*. Add formula at each leaf below current node. Do not tick the node.

### ∀xφ

### φ(t/x)

*γ*

Pick any closed term *t*. Add formula at each leaf below current node. Do not tick the node.

## ∀xφ ¬∃xφ

### φ(t/x) ¬φ(t/x)

Observe: ¬∃*xφ* ≡ ∀*x*¬*φ*

Quiz Time!

formulae below?

I ∀*x* ¬*p*(*x*)

I *H*(*a*) →*F*(*a*)

I ¬¬∃*y p*(*y*)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*))

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*))

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule) I *H*(*a*) →*F*(*a*)

I ¬¬∃*y p*(*y*)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*))

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*))

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*))

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*))

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*))

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*))

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*))

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*)) (*α* rule)

I *G*(*a*) →*H*(*a*)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*)) (*α* rule)

I *G*(*a*) →*H*(*a*) (*β* rule)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*)))

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*)) (*α* rule)

I *G*(*a*) →*H*(*a*) (*β* rule)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*))) (*α* rule)

I

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*)) (*α* rule)

I *G*(*a*) →*H*(*a*) (*β* rule)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*))) (*α* rule)

I (*γ* rule)

formulae below?

I ∀*x* ¬*p*(*x*) (*γ* rule)

I *H*(*a*) →*F*(*a*) (*β* rule)

I ¬¬∃*y p*(*y*) (*α* rule)

I ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) (*α* rule)

I ¬(∀*x*¬*q*(*x*) ∨∃*x*∀*y*¬(*x <y*)) (*α* rule)

I *G*(*a*) →*H*(*a*) (*β* rule)

I ¬¬(∀*x* (*G*(*x*) →*H*(*x*)) ∧∀*x* (*H*(*x*) →*F*(*x*)) ∧*G*(*a*) ∧¬∃*x* (*G*(*x*) ∧

*F*(*x*))) (*α* rule)

I (*γ* rule)

(*δ* rule)

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*))

I *G*(*a*)

I ¬*p*(*c*)

I ∀*x* (*G*(*x*) →*H*(*x*))

I ¬*H*(*a*)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*)

I ¬*p*(*c*)

I ∀*x* (*G*(*x*) →*H*(*x*))

I ¬*H*(*a*)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*)

I ∀*x* (*G*(*x*) →*H*(*x*))

I ¬*H*(*a*)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*))

I ¬*H*(*a*)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule) I ¬*H*(*a*)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule)

I ¬*H*(*a*) (none - literal)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*)) I ¬*G*(*a*)

I ¬(*G*(*a*) ∧*F*(*a*))

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule)

I ¬*H*(*a*) (none - literal)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*))

(*α* rule) I ¬*G*(*a*)

I

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule)

I ¬*H*(*a*) (none - literal)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*))

(*α* rule)

I ¬*G*(*a*) (none - literal)

I

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule)

I ¬*H*(*a*) (none - literal)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*))

(*α* rule)

I ¬*G*(*a*) (none - literal)

I (*β* rule)

formulae below?

I ∀*x* (*H*(*x*) →*F*(*x*)) (*γ* rule)

I *G*(*a*) (none - literal)

I ¬*p*(*c*) (none - literal)

I ∀*x* (*G*(*x*) →*H*(*x*)) (*γ* rule)

I ¬*H*(*a*) (none - literal)

I ∀*x* (*G*(*x*)→*H*(*x*))∧∀*x* (*H*(*x*)→*F*(*x*))∧*G*(*a*)∧¬∃*x* (*G*(*x*)∧*F*(*x*))

(*α* rule)

I ¬*G*(*a*) (none - literal)

I (*β* rule)

(*δ* rule)

1. ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*))

*α*(1)

1. ∀*x* ¬*p*(*x*)
2. ¬¬∃*y p*(*y*)

*α*(3)

1. ∃*y p*(*y*)

*δ*(4*,c*)

1. *p*(*c*)

*γ*(2*,c*)

1. ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) X

*α*(1)

1. ∀*x*¬*p*(*x*)
2. ¬¬∃*y p*(*y*)

*α*(3)

1. ∃*y p*(*y*)

*δ*(4*,c*)

1. *p*(*c*)

*γ*(2*,c*)

1. ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) X

*α*(1)

1. ∀*x*¬*p*(*x*)
2. ¬¬∃*y p*(*y*) X

*α*(3)

1. ∃*y p*(*y*)

*δ*(4*,c*)

1. *p*(*c*)

*γ*(2*,c*)

1. ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) X

*α*(1)

1. ∀*x*¬*p*(*x*)
2. ¬¬∃*y p*(*y*) X

*α*(3)

1. ∃*y p*(*y*) X

*δ*(4*,c*)

1. *p*(*c*)

*γ*(2*,c*)

* 1. ¬(∀*x* ¬*p*(*x*) →¬∃*y p*(*y*)) X

*α*(1)

* 1. ∀*x* ¬*p*(*x*) X
  2. ¬¬∃*y p*(*y*) X

*α*(3)

* 1. ∃*y p*(*y*) X

*δ*(4*,c*)

* 1. *p*(*c*)

*γ*(2*,c*)

* 1. ¬*p*(*c*)

⊕

Closed tableau — root formula is unsatisfiable, original formula is valid.



(4)∀*x*(*Hx* →*Fx*)

(5)*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

*γ*

(6

*,*

*a*

)

(8)¬*Ga β*(7) (9)¬*Fa*

⊕

(10)

*Ha*

→

*γ*

(4

*,*

*a*

)

*Fa*

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕



(4)∀*x*(*Hx* →*Fx*)

(5)*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

*γ*

(6

*,*

*a*

)

(8)¬*Ga β*(7) (9)¬*Fa*

⊕

(10)

*Ha*

→

*γ*

(4

*,*

*a*

)

*Fa*

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

*α*(2)

(3)∀*x*(*Gx* →*Hx*)

(4)∀*x*(*Hx* →*Fx*)

(5)*Ga*

(6)¬∃*x*(*Gx* ∧*Fx*)

*γ*(6*,a*)

(7)¬(*Ga*∧*Fa*)

(8)¬*Ga β*(7) (9)¬*Fa*

⊕

(10)

*Ha*

→

*γ*

(4

*,*

*a*

)

*Fa*

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

*α*(2)

(3)∀*x*(*Gx* →*Hx*)

(4)∀*x*(*Hx* →*Fx*)

(5)*Ga*

(6)¬∃*x*(*Gx* ∧*Fx*)

*γ*(6*,a*)

(7)¬(*Ga*∧*Fa*)

(8)¬*Ga β*(7) (9)¬*Fa*

⊕

(10)

*Ha*

→

*γ*

(4

*,*

*a*

)

*Fa*

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

*Fa*

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

*α*

(2)

*γ*

(6

*,*

*a*

)

(10)

*Ha*

→

*γ*

(4

*,*

*a*

)

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*Ha β*(10) (12)*Fa*

(11)

¬

*γ*

(3

*,*

*a*

)

⊕

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

X

(11)

¬

*Ha*

*β*

(10)

(12)

*Fa*

⊕

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*γ*

(3

*,*

*a*

)

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

X

(11)

¬

*Ha*

*β*

(10)

(12)

*Fa*

⊕

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*γ*

(3

*,*

*a*

)

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

X

(11)

¬

*Ha*

*β*

(10)

(12)

*Fa*

⊕

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*γ*

(3

*,*

*a*

)

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

X

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

X

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

X

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

X

(11)

¬

*Ha*

*β*

(10)

(12)

*Fa*

⊕

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*γ*

(3

*,*

*a*

)

(2)∀*x*(*Gx* →*Hx*)∧∀*x*(*Hx* →*Fx*)∧*Ga*∧¬∃*x*(*Gx* ∧*Fx*) X

(3)

∀

*x*

(

*Gx*

→

*Hx*

)

X

(4)

∀

*x*

(

*Hx*

→

*Fx*

)

X

(5)

*Ga*

(6)

¬∃

*x*

(

*Gx*

∧

*Fx*

)

X

(7)

¬

(

*Ga*

∧

*Fa*

)

X

(8)

¬

*Ga*

⊕

*β*

(7)

(9)

¬

*Fa*

(10)

*Ha*

→

*Fa*

X

(11)

¬

*Ha*

*β*

(10)

(12)

*Fa*

⊕

*α*

(2)

*γ*

(6

*,*

*a*

)

*γ*

(4

*,*

*a*

)

*γ*

(3

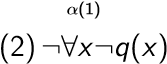
*,*

*a*

)

Example 5: Is formula ∀*x*¬*q*(*x*)∨∃*x*∀*y*¬(*x < y*) valid?

(1)¬(∀*x*¬*q*(*x*)∨∃*x*∀*y*¬(*x < y*)) X

 X

(3)¬∃*x*∀*y*¬(*x < y*)

*δ*(2*,c*)

(4)¬¬*q*(*c*) X

*α*(4)

(5)*q*(*c*)

*γ*(3*,c*)

(6)¬∀*y*¬(*c< y*) X

*δ*(6*,d*)

(7)¬¬(*c< d*) X

*α*(7)

(8)(*c< d*)

*γ*(3*,d*)

(9)¬∀*y*¬(*d< y*) X

*δ*(9*,e*)

(10)¬¬(*d< e*) X

*α*(10)

(11)(*d< e*)

*...*

# Alternative: tableaux as lists

Recall

literal *p,*¬*p*

*α* (*φ*1 ∧ *φ*2)*,*¬(*φ*1 ∨ *φ*2)*,*¬(*φ*1 → *φ*2)*,*¬¬*φ β* (*φ*1 ∨ *φ*2)*,*(*φ*1 → *φ*2)*,*¬(*φ*1 ∧ *φ*2)

# Formula expansions

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  |  | | --- | --- | --- | | *α* | *α*1 | *α*2 | | (*A* ∧ *B*) | *A* | *B* | | ¬(*A* ∨ *B*) | ¬*A* | ¬*B* | | ¬(*A* → *B*) | *A* | ¬*B* | | ¬¬*A* | *A* | — | | |  |  |  | | --- | --- | --- | | *β* | *β*1 | *β*2 | | (*A* ∨ *B*) | *A* | *B* | | (*A* → *B*) | ¬*A* | *B* | | ¬(*A* ∧ *B*) | ¬*A* | ¬*B* | |

# Propositional Tableaux

I A theory Σ is a set of formulas.

I If *p,*¬*p* ∈ Σ theory is contradictory, write *C*(Σ)

I If each formula in Σ is a literal, theory is fully expanded, write *Exp*(Σ)

I A tableau is a list of theories. Think of these as alternative theories.

# Tableau(*φ*)

Initialise *Tab* = [{*φ*}] while Not empty *Tab* do Σ = *Dequeue*(*Tab*) if *Exp*(Σ) and NOT *C*(Σ) then

Output SATISFIABLE

else

Pick non-literal *ψ* ∈ Σ switch (*ψ*) case *α*:

Σ = Σ[*α/*{*α*1*,α*2}], if NOT *C*(Σ) and Σ 6∈ *Tab* then enqueue Σ case *β*:

Σ1 = Σ[*β/β*1], if Σ1 6∈ *Tab* and NOT *C*(Σ1) then enqueue Σ1

Σ2 = Σ[*β/β*2], if Σ2 6∈ *Tab* and NOT *C*(Σ2) then enqueue Σ2 end switch

end if

end while

(Empty *Tab*) Output UNSATISFIABLE

# Predicate Tableaux

Under switch statement add cases *δ* and *γ*.

switch (*ψ*) case *δ* = ∃*xθ*(*x*):

Σ = Σ[∃*xθ*(*x*)*/θ*(*p*)] (new const *p*) case *δ* = ¬∀*xθ*(*x*): fill in case *γ* = ∀*xθ*(*x*): pick closed term *t* ∈ Σ, Σ = Σ ∪ {*θ*(*t*)}.

case *γ* = ¬∃*xθ*(*x*): fill in

end switch

If Σ 6∈ *Tab* and NOT *C*(Σ) enqueue Σ.

# Soundness of Propositional Tableau Algorithm

If *φ* is satisfiable then tableau for *φ* cannot close.

Proof.

Assume *v*(*φ*) = >. Prove by induction on number *n* of iterations of while statement that there is Σ ∈ *Tab* where *θ* ∈ Σ → *v*(*θ*) = >.

True for *n* = 0 by initialisation and assumption. Assume after *n* iterations that there is Σ ∈ *Tab* such that *θ* ∈ Σ → *v*(*θ*) = >.

If Σ is dequeued and *ψ* ∈ Σ is picked we have *v*(*ψ*) = > (by IH).

If *ψ* is an *α* then *v*(*α*1) = *v*(*α*2) = > so *θ* ∈ Σ → *v*(*θ*) = > still true. If *ψ* is a *β* then either *v*(*β*1) = > or *v*(*β*2) = >, so either *θ* ∈ Σ1 → *v*(*θ*) = > or *θ* ∈ Σ2 → *v*(*θ*) = >.

Result follows by induction over *n*.

# Soundness of Predicate Tableau Algorithm

Assume *φ* has a model.

Prove by induction on number of iterations of while loop that there is Σ ∈ *Tab* and *S,A* |= Σ (some *S,A*).

True for *n* = 0 by assumption.

Assume Σ ∈ *Tab* and *S,A* |= Σ.

When Σ is dequeued and *ψ* is picked, cases where *ψ* is *α* or *β* are same as propositional case, no change to *S,A* needed.

If *ψ* is *δ*, say *ψ* = ∃*xθ*(*x*) then *S,A* |= ∃*xθ*(*x*) (by IH).

So there is *A*0 ≡*x A, S,A*0 |= *θ*(*x*).

∃*xθ*(*x*) gets replaced by *θ*(*p*) in Σ (new constant *p*).

Let *S*0 be same as *S* except *I*(*p*) = *A*0(*x*). Then *S*0*,A* |= Σ. If *ψ* is a *γ*, say *ψ* = ∀*xθ*(*x*) then *S,A* |= ∀*xθ*(*x*) (I.H.) Follows that *S,A* |= *θ*(*t*) for any closed term *t*.

∀*xθ*(*x*) is replaced by *θ*(*t*) (some closed term *t*) in Σ.

*S,A* |= Σ still true (no need to change *S,A* in this case).

# Ancestors

If Σ ∈ *Tab* is dequeued and Σ1 (Σ2) are enqueued, then Σ is parent of Σ1 (Σ2)

*P*(Σ) = Σ0 if Σ0 is the parent of Σ

*P*0(Σ) = Σ

*Pn*+1(Σ) = *P*(*Pn*(Σ))

Say Σ is ancestor of Σ0 if there is *n* ≥ 0 and *Pn*(Σ0) = Σ.

Initial tableau [{*φ*}] and {*φ*} is ancestor of every theory in the tableau.

# Completeness of Propositional Tableau Algorithm

Start tableau with *φ*. If algorithm outputs SATISFIABLE then there is *v* : *Props* → {>*,*⊥} such that *v*(*φ*) = >, so *φ* is satisfiable.

Proof.

Output SATISFIABLE ⇒ Σ ∈ *Tab* dequeued ∧ *Exp*(Σ) ∧ ¬*C*(Σ) Define *v* by *v*(*p*) = > ⇐⇒ *p* ∈ Σ.

Prove, by induction over *n*, that *ψ* ∈ *Pn*(Γ) ⇒ *v*(*ψ*) = >.

True for *n* = 0, by def. of *v*.

IH *ψ* ∈ *Pn*(Σ) ⇒ *v*(*ψ*) = >.

IS. Suppose *θ* ∈ *Pn*+1(Σ).

Either *θ* ∈ *Pn*(Σ) (so *v*(*θ*) = > by IH), or *θ* was expanded and replaced in *Pn*(Σ). *θ* = *α* ⇒ *α*1*,α*2 ∈ *Pn*(Σ), by IH *v*(*α*1) = *v*(*α*2) = >, so *v*(*α*) = >. *θ* = *β* ⇒ either *β*1 ∈ *Pn*(Σ) or *β*2 ∈ *Pn*(Σ).

By IH, either *v*(*β*1) = > or *v*(*β*2) = >.

Either way, *v*(*β*) = >.

# Termination of propositional tableau algorithm

I When running tableau algorithm for *φ*, only new theories are enqued.

I When running tableau algorithm for *φ*, only new theories are enqued.

I Let *X* be set of subformulas of *φ* and single negations of subformulas of *φ* — at most 2|*φ*| of these.

I When running tableau algorithm for *φ*, only new theories are enqued.

I Let *X* be set of subformulas of *φ* and single negations of subformulas of *φ* — at most 2|*φ*| of these.

I A theory is a subset of *X* — at most 2|*φ*| of these.

I When running tableau algorithm for *φ*, only new theories are enqued.

I Let *X* be set of subformulas of *φ* and single negations of subformulas of *φ* — at most 2|*φ*| of these.

I A theory is a subset of *X* — at most 2|*φ*| of these.

I Algorithm terminates in 2|*φ*| steps (at most).

Completeness of algorithm for predicate tableau

Soon.

# Herbrand Structures

A closed term *t* is built up from constants and function symbols only — no variables.

A Herbrand structure *H* = (*D,I*) has

Domain

*D* = {closed terms}

Interpretation *I* = (*Ic,If ,Ip*).

*Ic*(*c*) = *c*

*If* (*f n*) : (*d*1*,...,dn*) 7→ *f n*(*d*1*,...,dn*)

*Ip* can be chosen freely.

It follows, for any closed term *t*, that

[*t*]*H,A* = *t*

# Herbrand Theorem

Let *L* be a language with ∞ many constant symbols (and no equality predicate in this version of the theorem).

If *φ* is satisfiable (i.e. *S,A* |= *φ*, some *S* some *A*) then *φ* is satisfiable in a Herbrand model *H*, i.e. *H,A* |= *φ* (some *A*).

# Fairness

Suppose you have several (countably many) processes *P*1*,P*2*,...,Pk,...* and each of them is waiting for some input. It might be that when you give *Pi* some input, it creates a new process *Pk*+1, so the list can grow, but it will always be countable. In what order should you supply inputs to the various processes? You could simply supply input to *P*1 again and again, but that would be unfair to all the other processes. In a fair schedule, if any process *Pi* is waiting for input at time *t* then eventually (at some time *t*0 *> t*) *Pi* will get some input. If processes are always waiting for input, then each process will get input infinitely often.

Since the total number of requests for input is countable, it is possible to find a fair schedule.

# Completeness of algorithm for predicate tableau

If tableau for *φ* never closes and expanded by a fair schedule then *φ* is satisfiable.

Proof: If tableau never closes, by K¨onig’s tree lemma there is a sequence

Σ0*,*Σ1*,*Σ2*,...* ∈ *Tab* where Σ*n* = *P*(Σ*n*+1). Let Σ = S*n<*∞ Σ*n*.

Since fair schedule was used,

*α* ∈ Σ ⇒ *α*1 ∈ Σ and *α*2 ∈ Σ *β* ∈ Σ ⇒ *β*1 ∈ Σ or *β*2 ∈ Σ

∃*xθ*(*x*) ∈ Σ ⇒ *θ*(*p*) ∈ Σ(some *p*)

¬∀*xθ*(*x*) ∈ Σ ⇒ ¬*θ*(*p*) ∈ Σ (some *p*)

∀*xθ*(*x*) ∈ Σ ⇒ *θ*(*t*) ∈ Σ (all closed terms *t*)

¬∃*xθ*(*x*) ⇒

# Completeness, continued

Let *H* be Herbrandt structure, base {closed terms of Σ}, *I*(*t*) = *t* (closed term *t*) and

(*t*0*,t*1*,...,tk*−1) ∈ *I*(*Rn*) ⇐⇒ *Rn*(*t*0*,...,tn*−1) ∈ Σ. Prove by structured formula induction that *θ* ∈ Σ ⇒ *H* |= *θ* and

¬*θ* ∈ Σ ⇒ *H* |= ¬*θ*.

Hence *H* |= *φ*.

# Other proof systems

Tableau Tests satisfiability, not validity. Easy to implement.

Tableau Tests satisfiability, not validity. Easy to implement.

Axiomatic Systems Easy to define, very hard to use.

Tableau Tests satisfiability, not validity. Easy to implement.

Axiomatic Systems Easy to define, very hard to use.

Resolution Theorem Provers Checks validity. Prolog.

Tableau Tests satisfiability, not validity. Easy to implement.

Axiomatic Systems Easy to define, very hard to use.

Resolution Theorem Provers Checks validity. Prolog.

Natural Deduction Easier to read the proofs.

Tableau Tests satisfiability, not validity. Easy to implement.

Axiomatic Systems Easy to define, very hard to use.

Resolution Theorem Provers Checks validity. Prolog.

Natural Deduction Easier to read the proofs.

Truth Tables Very simple idea. Takes exponential time. Does not work for most logics (other than propositional logic).

# Theorem Proving for Predicate Logic. Axiomatic Proofs

Take axioms schemas for propositional logic. Quantifier Axioms:

1. (∀*x*¬*A* ↔ ¬∃*xA*)
2. (∀*xA*(*x*) → *A*(*t/x*)) if *t* is substitutable for *x* in *A*. X. (∀*x*(*A* → *B*) → (∀*xA* → ∀*xB*)).

Equality Axioms:

1. (*x* = *x*)
2. ((*x* = *y*) → (*t*(*x*) = *t*(*y/x*)))
3. ((*x* = *y*) → (*A*(*x*) → *A*(*y/x*))) if *y* is substitutable for *x* in *A*.

An instance of any of the axioms above is obtained by replacing *A,B,C* etc. by arbitrary formulas.

# Inference Rules

Modus Ponens

|  |  |
| --- | --- |
| Universal Generalisation | *A,*(*A* → *B*)  *B* |

*A*(*x*)

∀*xA*(*x*)

(N.B. *A*(*x*) → ∀*xA*(*x*) is not an axiom, as it is not valid. Universal generalisation says that if *A*(*x*) is valid then ∀*xA*(*x*) is also valid.

This rule is sound.)

# Proofs

A proof of *φ* is a finite sequence

## φ0,φ1,φ2,...,φn = φ

such that, for each *i* ≤ *n*, either

I *φi* is an instance of one of the axioms or

I *φi* is obtained from *φj* (and maybe *φk*) where *j,k < i*, by an inference rule.

Write

` *φ*

in this case.

# Proving from hypotheses

So far, this is all to do with validity over arbitrary models. If you want to find validities in a particular model, or a particularly type of model, then you can add hypotheses.

These hypotheses are formulas which are valid in the type of formula you want, and they define it. E.g. Linearly Ordered Models Hypotheses:

∀*x*∀*y*(*x < y* ∨ *y < x* ∨ *x* = *y*)

∀*x*¬(*x < x*)

∀*x*∀*y*∀*z*((*x < y* ∧ *y < z*) → *x < z*)

# Proofs with hypotheses

Let Γ be a set of hypotheses. Write

Γ ` *φ*

if there is a sequence *φ*0*,φ*1*,...,φn* = *φ*

such that for each *i* ≤ *n* either

I *φi* is an axiom,

I *φi* is obtained from *φj* (*φk*) (some *j,k < i*) by an inference rule, or I *φi* ∈ Γ.

# Example Proof using Hypotheses

Linear Order ` ∀*x*∀*y*¬(*x < y* ∧ *y < x*)

Proof

I ∀*x*∀*y*∀*z*((*x < y* ∧ *y < z*) → *x < z*) (Hypothesis)

I ∀*x*∀*y*∀*z*((*x < y* ∧ *y < z*) → (*x < z*)) →

((*x < y* ∧ *y < x*) → *x < x*)(Ax. IX) I ((*x < y* ∧ *y < x*) → *x < x*) (Modus Ponens)

I ∀*x*¬(*x < x*) (Hypothesis)

I ((*x < y* ∧ *y < x*) → *x < x*) → (¬(*x < x*) → ¬(*x < y* ∧ *y < x*))

(Ax. III)

I (¬(*x < x*) → ¬(*x < y* ∧ *y < x*))(Modus Ponens)

I ¬(*x < x*) (Ax. IX)

I ¬(*x < y* ∧ *y < x*)(Modus Ponens)

I ∀*x*∀*y*¬(*x < y* ∧ *y < x*) (Universal Generalisation)

# Entailment

Let Γ be a set of formulas and let *S* be an L-structure. Write

*S* |= Γ

if *S* |= *φ* for each *φ* ∈ Γ (say “*S* is a model of Γ) and

Γ |= *φ*

if every model of Γ is a model of *φ* (i.e. *S* |= Γ ⇒ *S* |= *φ*).

# Strong Completeness

Γ ` *φ* ⇐⇒ Γ |= *φ*

Corollary

*There is an enumeration of the valid formulas.*

# Tableau Summary

I Tableau method is sound and complete for first order logic (this is, essentially, G¨odel’s completeness theorem).

I If *φ* is not satisfiable its tableau will close finitely, if fair sequence is used (completeness).

I If *φ* is satisfiable its tableau will never close (soundness).

I But a tableau construction may never terminate.

# Recursive Languages

A language *L* is just a set of strings over some finite alphabet Σ. *L* is recursive if there is a computer program that takes an arbitrary string *s* ∈ Σ∗ as an input and outputs

“yes” if *s* ∈ *L*

“no” otherwise

The program must be guaranteed to terminate, for any *s* ∈ Σ∗. The set of all formulas of first order logic is a recursive set (a parsing program decides if a string is a well formed formula). The valid statements of first order logic form a language, but this language is not recursive (not decidable).

# Recursively Enumerable Languages

A language *L* is recursively enumerable (r.e.) if there is a computer program that outputs strings from *L*, only strings from *L*, and will eventually output any given string from *L*.

The valid statements of first-order logic form a recursively enumerable language.

First Order Logic is r.e.

Let *φ*0*,φ*1*,...* be an enumeration of all formulas.

For (*i* = 0*,i* + +*,*forever)

{ Start new tableau *Ti* with ¬*φi* at root; For each *j < i*

{ expand *Tj* once, using a fair schedule; If *Tj* becomes closed, output “*φj* is valid”; }

}

Note: for any formula *φk*, if *φk* is valid then eventually *Tk* will close and the program will output *φk* (completeness, though you do not know how long this will take).

If *φk* is not valid then *Tk* will never close (soundness).

So the program only outputs valid formulas, and any given valid formula will eventually get output.

# Recursive and r.e. languages

The set of formulas of first-order logic is a recursive set.

The set of valid formulas of FOL is not recursive, but it is r.e.

The set of true statements of arithmetic is not even r.e.

This last statement is G¨odel’s incompleteness theorem.

# Proving from Assumptions

Suppose you want to prove that *φ* is valid in a particular model, or type of model (e. g. linear order).

Write down assumptions Σ that define this type of model. E.g.

 ∀*x*∀*y*(*x* = *y* ∨ *x < y* ∨ *y < x*)*,* 

 

Σ = ∀*x*∀*y*∀*z*((*x < y* ∧ *y < z*) → *x < z*)*,*

 ∀*x*¬(*x < x*) 

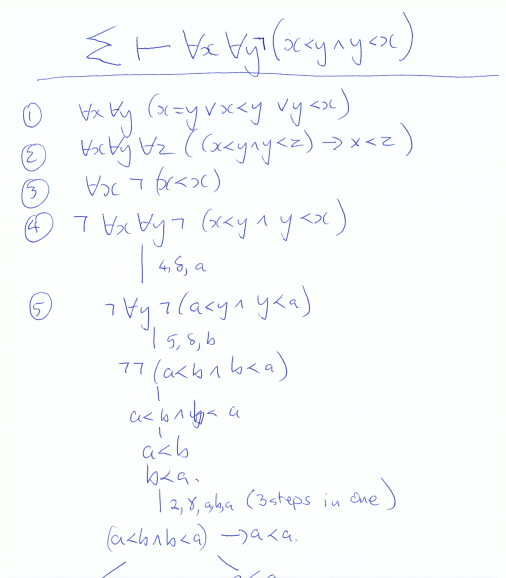
New rule: you can add any assumption in Σ at leaf of tableau at any time. A proof of *φ* using Σ is a closed tableau for ¬*φ*, but you can use assumptions to help you close the tableau. Write

Σ ` *φ*

# Tableau with assumptions example (TAE)

Σ = {∀*x*¬(*x < x*)*,* ∀*x*∀*y*(*x < y*∨*y < x*∨*x* = *y*)*,*∀*x*∀*y*∀*z*((*x < y*∧*y < z*) → *x <*

Σ ` ∀*x*∀*y*¬(*x < y* ∧ *y < x*)

Tableau for negated formula

# Entailment and Strong Completeness

Recall

Σ |= *φ* means *S* |= Σ ⇒ *S* |= *φ*

i.e. every model of Σ is also a model of *φ*. Strong Completeness

Σ |= *φ* ⇐⇒ Σ ` *φ*

# Compactness

If

Σ ` *φ*

then

Σ0 ` *φ*

for some finite subset Σ0 of Σ.

Inconsistency, Compactness, Completeness.

Σ is inconsistent if Σ ` (*p* ∧ ¬*p*)

If Σ is inconsistent then, by compactness, Σ0 is inconsistent, for some finite subset Σ0 of Σ.

By strong completeness theorem, every consistent set has a model. Hence, compactness says that if every finite subset of Σ has a model then there is a model for the whole of Σ.

# First Order Logic Cannot Define Connectedness

Language

*C* = {*c,d*}

*F* = ∅

*P* = {=*,E*} (both binary)

Suppose, for contradiction, that

*G* |= Σ ⇐⇒ *G* is connected

Let

*φ*1(*x,y*) = *E*(*x,y*) *φn*+1(*x,y*) = ∃*z*(*φn*(*x,z*) ∧ *E*(*z,y*))

“there is a path of length *n* + 1 from *x* to *y*”.

Consider

Σ ∪ {¬*φ*1(*c,d*)*,*¬*φ*2(*c,d*)*,...*}

Every finite subset has a model (what model?). By compactness, the whole set has a model, say *G*,

*G* |= Σ ∪ {¬*φn*(*c,d*) : *n* = 1*,*2*,*3*,...*}

*G* is therefore connected (since a model of Σ), but there is no path from *c* to *d* — a contradiction.

# Compactness theorem and non-standard analysis

Let

Σ = {all valid statements about N}

in a language with constants 0*,*1*,*2*,...* functions +*,*× and predicate =.

E.g. 2 + 2 = 4 ∈ Σ.

Also ∀*x*∀*y*(*x* × *y* = *y* × *x*) ∈ Σ.

Let *ω* be another constant symbol.

Every finite subset of

Σ+ = Σ ∪ {*ω* 6= 0*,ω* 6= 1*,ω* 6= 2*,...,ω* 6= *n,...*}

has a model (what model?).

Therefore Σ+ has a model.

# Non-standard real analysis

Let *L* be similar but with a constant for every real number. Let

Σ = {all valid statements about R}

and

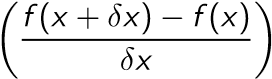
Σ+ = Σ ∪ {*α > r* : *r* ∈ R}

Every finite subset of Σ+ has a model (just interpret *α* as a sufficiently big real number), therefore Σ+ has a model *M*. Then [*α*]*M* is an “infinitely big” real number and [*α*1]*M* is and “infinitesimally small” positive real number.

Can do calculus perfectly rigorously in this way. Can show that

∀*x*((|*x*| *< r*) → (*x* = *St*(*x*) + *Inf* (*x*)))

where *r* is a constant for any positive real, *St*(*x*) is a “standard real” and *Inf* (*x*) is an “infinitesimal real”. Then let

*f* 0(*x*) = *St*

where *x* is any standard real and *δx* is any infinitesimal, provided this does not depend on the choice of *δx*.

# Paradoxes

I Liar Paradox, “All Cretans are liars” I Liar Paradox, “All Cretans are liars”

I Russell’s paradox. *R* = {*S* : *S* 6∈ *S*}. Does *R* ∈ *R*?

I Liar Paradox, “All Cretans are liars”

I Russell’s paradox. *R* = {*S* : *S* 6∈ *S*}. Does *R* ∈ *R*? I Berry’s paradox. Smallest natural number that cannot be defined uniquely by up to 80 characters.

# Go¨del’s Incompleteness Theorem

Consider true statements of arithmetic.

|  |  |  |
| --- | --- | --- |
| *C* | = | {0*,*1*,*2*,...*} |
| *F* | = | {+*,*×} |
| *P* | = | {=*,<*} |

Theorem (Go¨del, 1931)

*If S is any r.e. set of L-sentences then either*

I *There is a statement φ which is true in arithmetic (*N*) but φ* 6∈ *S (incompleteness), or*

I *There is a statement φ which is false in arithmetic and φ* ∈ *S (inconsistency).*

We will prove a slightly weaker result: if Γ is any finite set of axioms in this language then either Γ ` ⊥, i.e. Γ is inconsistent, or there is a formula *φ* which happens to be true in N and yet Γ 6` *φ* (incompleteness).

# Proof Sketch

Idea: every character coded as a number

|  |  |  |  |
| --- | --- | --- | --- |
| char | code |  |  |
| *p* | 11 | *x* | 31 |
| ( | 21 | ) | 22 |
| ∧ | 411 | ∨ | 42 |
| ¬ | 43 |  |  |
| ∀  *...* | 44 | ∃ | 45 |

Every string is coded using 0 as a delimiter, e.g. *p*(*x*) has code 11 0 21 0 31 0 22.

So can code a formula as a number.

Also, given a code like this, we can recover syntactic information. E.g. let *n* be a code. The statement “the last character of the formula with code *n* is a right bracket” can be expressed by the formula

∃*z n* = 1000 × *z* + 212

This code number is called the “G¨odel number” of the formula.

# Go¨del Coding

Can convert formula to code and code back to formula. Can write a first order formula *φ*(*n*) that is true iff *n* is the G¨odel number of a formula.

Similarly, every proof can be represented as a string using 00 as a delimeter, so every proof has a G¨odel number.

Let *G,F,P* be coding and decoding functions, so if *φ* is a formula and *φ*¯ is a proof then *G*(*φ*) and *G*(*φ*¯) are their codes numbers. If *n* ∈ N then *F*(*n*) is the formula *φ* (if any) such that *G*(*φ*) = *n* and *P*(*n*) is the proof *φ*¯ such that *G*(*φ*¯) = *n*.

# The proof

Can write formulas

|  |  |  |
| --- | --- | --- |
| *µ*(*n,m*) | = | *P*(*n*) is a proof of *F*(*m*) |
| *λ*(*n*)  Let | = | *F*(*n*) is a formula with one free variable, *x* |

*A*0(*x*)*,A*1(*x*)*,A*2(*x*)*,...*

be an enumeration of all the formulas with one free variable *x*. If *F*(*m*) has just *x* as a free variable then *F*(*m*) = *Ak*(*x*) (some *k*). Can write *µ*(*n,k,q*) = (*P*(*n*) is a proof of *Ak*(*q*))

Consider

¬∃*nµ*(*n,x,x*) This is a formula with one free variable. So there is some *n*0 such that

*An*0(*x*) = ¬∃*nφ*(*n,x,x*)

We have N |= *An*0(*m*) iff “there is no proof of *Am*(*m*)”.

Finally, consider

*An*0(*n*0)

We have

N |= *An*0(*n*0) ⇐⇒ there is no proof of *An*0(*n*0) !

If N |= *An*0(*n*0) then no proof of *An*0(*n*0) (incompleteness).

If N 6|= *An*0(*n*0) then there is a proof of *An*0(*n*0). (inconsistency).

# Other Logics

1. Second order, third order, ... higher order logic (no completeness, so not even r.e.).
2. Fixpoint logic.
3. Modal Logic.
4. Temporal Logic.

5.

Dynamic Logic 



Intuitionistic Logic funny def. of →

Linear Logic 

1. Algebraic Logic.
2. Horn Clause Logic (Prolog) — satisfiability for propositional case is in P.

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

|  |  |  |
| --- | --- | --- |
|  |  |  |
|  |  |  |
|  |  |  |

# Modal Operators: ♦ and

♦*p* could mean

I “*p* is possible”

I “*p* might happen”

I “*p* will happen (at some time in the future)”

I “I think *p* could be true”

*p* = ¬♦¬*p*. For each of the above, what does *p* mean?

# Syntax

*prop* ::=*p*|*q*|*r*|*...*



e.g.

(♦*g* → ♦*g*)

# Frames

F = (*W,R*) where *R* ⊆ *W* × *W* (a directed graph).

Valuation:

*V* : *prop* → *℘*(*W*) e.g.

*V*(*p*) = {1*,*3*,*5}

*V*(*q*) = {1*,*2}

*V*(*r*) = ∅

# Semantics

Model M = (*W,R,V*). Let *v* ∈ *W*.

|  |  |
| --- | --- |
| M*,v* |=*p* | ⇐⇒ *v* ∈ *V*(*p*) |
| M*,v* |=¬*φ* | ⇐⇒ M*,v* 6|= *φ* |
| M*,v* |=(*φ* ∧ *θ*) | ⇐⇒ M*,v* |= *φ* and M*,v,*|= *θ* |
| M*,v* |=♦*φ* | ⇐⇒ there is *w* ∈ *W* (*v,w*) ∈ *R* and M*,w* |= *φ* |

 for all *w* ∈ *W* if (*v,w*) ∈ *R* then M*,w* |= *φ*

# Validity

I Valid in a model

(*W,R,V*) |= *φ* ⇐⇒ for all *v* ∈ *W* (*W,R,V*)*,v* |= *φ*

I Valid in a frame

(*W,R*) |= *φ* ⇐⇒ for all valuations *V* (*W,R,V*) |= *φ*

I Valid over a class of frames

K |= *φ* ⇐⇒ for all frames F ∈ K F |= *φ*

Summary

Summary

I Finite State Machines

# Summary

I Finite State Machines

I Non-deterministic FSMs

# Summary

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions

# Summary

I Finite State Machines

I Non-deterministic FSMs I Regular languages, regular expressions I Kleene’s theorem.

# Summary

I Finite State Machines

I Non-deterministic FSMs I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Tableaus for propositional logic.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Tableaus for propositional logic.

I Tableaus for first order logic.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Tableaus for propositional logic.

I Tableaus for first order logic.

I Fairness.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Tableaus for propositional logic.

I Tableaus for first order logic. I Fairness.

I Soundness and completeness of tableau.

I Finite State Machines

I Non-deterministic FSMs

I Regular languages, regular expressions I Kleene’s theorem.

I Propositional and first-order logic.

I Validity in a first order structure. Validity |= over all structures.

I Satisfiability in a strucutre. Satisfiability in some structure.

I Tableaus for propositional logic.

I Tableaus for first order logic. I Fairness.

I Soundness and completeness of tableau.

I Compactness