

# G22.3033-003 Logic and Verification

## Instructor:

Clark W. Barrett  
[barrett@cs.nyu.edu](mailto:barrett@cs.nyu.edu)

1

## Course Goals

- **Prerequisites:**

Logic in Computer Science (taught last semester) or equivalent knowledge of first-order logic

- **Contents of the Course**

This course is about automated reasoning, and specifically about decision procedures for decidable fragments of first-order logic. We will cover a variety of problem domains and techniques for creating decision procedures and will look at several important applications.

3

## Outline

- Goals and Organization
- History and Motivation
- Propositional and First Order Logic
- Computability and Decidability
- Proofs and Theories

### Sources:

Siekmann and Wrightson Ed. *Automation of Reasoning*, vol. 1 and 2. Springer-Verlag, 1983.

Shankar et al. *CS359: Little Engines of Proof*. Lecture notes, Fall 2003. <http://www.csl.sri.com/users/shankar/LEP.html>.

Enderton, Herbert B. *A Mathematical Introduction to Logic, Second Edition*. Academic Press, 2001.

Hodges, W. *A Shorter Model Theory*. Cambridge Press, 1997.

Cutland, N. J. *Computability*. Cambridge Press, 1980.

2

## Course Information

### Webpage:

<http://cs.nyu.edu/courses/spring04/G22.3033-003/index.htm>

### Sources:

The sources for this class will be the lecture notes and various research papers. The papers we cover in detail will be available online or as handouts.

### Assignments:

There will be periodic assignments which will generally be due the following week (unless otherwise indicated).

### Projects:

In addition to assignments there will be a final project which will require a significant amount of programming effort.

### Grading:

Assignments: 50%, Project: 50%

4

## Why Automated Reasoning?

From very early on, philosophers have dreamed of machines that can reason.

*Leibniz* (1646-1716), who with *Newton* generally shares credit for inventing Calculus, first proposed the ambitious goal of mechanizing the process of human reasoning, saying,

Once this is done, then when a controversy arises, disputation will no more be needed between two philosophers than between two computers. It will suffice that, pen in hand, they sit down to their abacus and (calling in a friend if they so wish) say to each other: *let us calculate*.

5

## Why Automated Reasoning?

Mathematicians have long understood the importance of proofs as an assurance of the correctness of a logical argument.

Many modern proofs in mathematics are very difficult, and could benefit immensely from automated proof assistance.

In 1998, *Tom Hales* gave a proof of a famous unsolved problem known as the **Kepler Conjecture**. The proof was so long and complicated that it defied all human efforts to check it.

As a result, *Hales* began the **Flyspeck** project, an effort to automate the proof the Kepler conjecture (see <http://www.math.pitt.edu/thales/flyspeck/>).

More recently, the need for proofs in computer science applications has become urgent. Proofs provide assurances of correctness that cannot be obtained using simulation and testing.

Proof-based verification techniques are now standard in hardware design and are becoming more and more prevalent in software design as well.

7

## Why Automated Reasoning?

In 1945, *Dr. Vannevar Bush*, director of the Office of Scientific Research and Development, wrote the following:

Logic can become enormously difficult, and it would undoubtedly be well to produce more assurance in its use. ... We may some day click off arguments on a machine with the same assurance that we now enter sales on a cash register.

From **As We May Think**,  
<http://www.theatlantic.com/unbound/flashbks/computer/bushf.htm>

6

## History

Logic itself dates back to the 4th century BC with *Aristotle's* attempt to analyze sound reasoning.

*Leibniz* (17th century) had great ambitions for symbolic logic, but did not get very far himself.

The subject really blossomed in the 19th century with the work of *Boole*, *De Morgan*, *Dedekind*, *Frege*, *Peano*, *Pierce*, *Cantor*, and others.

The 20th century brought great minds like *Hilbert*, *Skolem*, *Herbrand*, *Gödel*, *Gentzen*, and *Church* who laid the foundation for modern mathematical logic.

8

## History

The very first computer-generated mathematical proof was generated here at New York University using a program written by [Martin Davis](#) in 1954. The program was an implementation of a decision procedure for Presburger Arithmetic.

Its great triumph was to prove that the sum of two even numbers is even.

In 1957, [Newell](#), [Shaw](#), and [Simon](#), researchers at RAND created a program called the Logic Theory Machine (LT).

The research...is aimed at understanding the complex processes (heuristics) that are effective in problem-solving...we are not interested in methods that guarantee solutions...Rather, we wish to understand how a mathematician, for example, is able to prove a theorem even though he does not know when he starts how, or if, he is going to succeed.

LT introduced many important ideas such as proof-trees, subgoals, and an early version of unification, and was successfully able to verify theorems from [Russell](#) and [Whitehead's Principia Mathematica](#).

9

## History

In 1965, [Robinson](#) proved that the simple resolution rule was a sound and complete inference procedure for first-order logic. Much of the work in automated deduction since then has been based on this rule.

However, for a given first-order theory, resolution may do poorly compared with domain-specific decision procedures.

Wang observed:

That proof procedures for elementary logic can be mechanized is familiar. In practice, however, were we slavishly to follow these procedures without further refinements, we should encounter a prohibitively expensive element. ...In this way we are led to a closer study of reduction procedures and of decision procedures for special domains.

In this course, we will look at some of these decision procedures for special domains, with an eye towards efficiency and practicality.

First, we will briefly review some of the relevant notions from logic.

11

## History

In 1960, [Hao Wang](#) published a paper reporting that many of the LT proofs as well as others from Russell and Whitehead were *easily* decidable. He was able to prove hundreds of these theorems in minutes.

The most interesting lesson from these results is perhaps that even in a fairly rich domain, the theorems actually proved are mostly ones which call on a very small portion of the available resources of the domain.

### Wang vs Newell-Shaw-Simon

On the contrasting approaches of Wang and Newell et al., Martin Davis wrote,

The controversy referred to may be succinctly characterized as being between the two slogans: "Simulate people" and "Use mathematical logic". Although this controversy has generated much heat, there has never been much doubt among serious workers in the field that both streams of ideas were important...Thus as early as 1961 Minsky remarked:

it seems clear that a program to solve real mathematical problems will have to combine the mathematical sophistication of Wang with the heuristic sophistication of Newell, Shaw and Simon.

10

## Logic

A formal logic is defined by its [syntax](#) and [semantics](#).

### Syntax

- An [alphabet](#) is a set of symbols.
- A finite sequence of these symbols is called an [expression](#).
- A set of rules defines the [well-formed](#) expressions.

### Semantics

- Gives meaning to well-formed expressions
- Formal notions of induction and recursion are required to provide a rigorous semantics.

12

## Propositional Logic: Syntax

### Alphabet

(	Left parenthesis	Begin group
)	Right parenthesis	End group
$\neg$	Negation symbol	English: not
$\wedge$	Conjunction symbol	English: and
$\vee$	Disjunction symbol	English: or (inclusive)
$\rightarrow$	Conditional symbol	English: if, then
$\leftrightarrow$	Bi-conditional symbol	English: if and only if
$A_1$	First propositional symbol	
$A_2$	Second propositional symbol	
...		
$A_n$	$n$ th propositional symbol	
...		

13

## Propositional Logic: Semantics

Intuitively, given a *wff*  $\alpha$  and a value (either *true* or *false*) for each propositional symbol in  $\alpha$ , we can determine the value of  $\alpha$ .

Let  $v$  be a function from  $B$  to  $\{0, 1\}$ , where  $0$  represents *false* and  $1$  represents *true*. Recall that in the inductive definition of *wff*'s,  $B$  contains the propositional symbols.

Now, we define  $\bar{v}$ , a function from  $W$  to  $\{0, 1\}$  as follows

- For each propositional symbol  $A_i$ ,  $\bar{v}(A_i) = v(A_i)$ .
- $\bar{v}(\mathcal{E}_{\neg}(\alpha)) = 1 - \bar{v}(\alpha)$
- $\bar{v}(\mathcal{E}_{\wedge}(\alpha, \beta)) = \min(\bar{v}(\alpha), \bar{v}(\beta))$
- $\bar{v}(\mathcal{E}_{\vee}(\alpha, \beta)) = \max(\bar{v}(\alpha), \bar{v}(\beta))$
- $\bar{v}(\mathcal{E}_{\rightarrow}(\alpha, \beta)) = \max(1 - \bar{v}(\alpha), \bar{v}(\beta))$
- $\bar{v}(\mathcal{E}_{\leftrightarrow}(\alpha, \beta)) = 1 - |\bar{v}(\alpha) - \bar{v}(\beta)|$

The recursion theorem guarantees that  $\bar{v}$  is well-defined, as long as the *wff*s are freely generated, which they are since each operation in  $F$  is one-to-one and has a range disjoint from the other operations in  $F$  and from  $B$ .

15

## Propositional Logic: Syntax

We use a formal inductive definition to define the set of well-formed formulas in propositional logic.

- $U =$  the set of all expressions.
- $B =$  the set of expressions consisting of a single propositional symbol.
- $F =$  the set of formula-building operations:
  - $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$
  - $\mathcal{E}_{\wedge}(\alpha, \beta) = (\alpha \wedge \beta)$
  - $\mathcal{E}_{\vee}(\alpha, \beta) = (\alpha \vee \beta)$
  - $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$
  - $\mathcal{E}_{\leftrightarrow}(\alpha, \beta) = (\alpha \leftrightarrow \beta)$

The set of well-formed formulas is the set of all expressions generated by  $F$  from  $B$ .

14

## Definitions

If  $\alpha$  is a *wff*, then a truth assignment  $v$  *satisfies*  $\alpha$  if  $\bar{v}(\alpha) = 1$ .

A *wff*  $\alpha$  is *satisfiable* if there exists some truth assignment  $v$  which satisfies  $\alpha$ .

Suppose  $\Sigma$  is a set of *wff*'s. Then  $\Sigma$  *tautologically implies*  $\alpha$ ,  $\Sigma \models \alpha$ , if every truth assignment which satisfies each formula in  $\Sigma$  also satisfies  $\alpha$ .

Particular cases:

- If  $\emptyset \models \alpha$ , then we say  $\alpha$  is a *tautology* or  $\alpha$  is *valid* and write  $\models \alpha$ .
- If  $\Sigma$  is *unsatisfiable*, then  $\Sigma \models \alpha$  for every *wff*  $\alpha$ .
- If  $\alpha \models \beta$  (shorthand for  $\{\alpha\} \models \beta$ ) and  $\beta \models \alpha$ , then  $\alpha$  and  $\beta$  are *tautologically equivalent*.
- $\Sigma \models \alpha$  if and only if  $\bigwedge(\Sigma) \rightarrow \alpha$  is valid.
- Satisfiability and validity are dual notions:  $\alpha$  is unsatisfiable if and only if  $\neg\alpha$  is valid.

16-g

## Truth Tables

Truth tables can be used to calculate all possible values of  $\bar{v}$  for a given *wff*: We associate a column with each propositional symbol and a column with each propositional connective. There is a row for each possible truth assignment to the propositional connectives.

A <sub>1</sub>	A <sub>2</sub>	A <sub>3</sub>	(A <sub>1</sub> ∨ (A <sub>2</sub> ∧ ¬A <sub>3</sub> ))
1	1	1	1
1	1	0	1
1	0	1	1
1	0	0	1
0	1	1	0
0	1	0	0
0	0	1	0
0	0	0	0

17-c

## Some tautologies

**Associative and Commutative laws for  $\wedge, \vee, \leftrightarrow$**

**Distributive Laws**

- $(A \wedge (B \vee C)) \leftrightarrow ((A \wedge B) \vee (A \wedge C))$ .
- $(A \vee (B \wedge C)) \leftrightarrow ((A \vee B) \wedge (A \vee C))$ .

**Negation**

- $\neg\neg A \leftrightarrow A$
- $\neg(A \rightarrow B) \leftrightarrow (A \wedge \neg B)$
- $\neg(A \leftrightarrow B) \leftrightarrow ((A \wedge \neg B) \vee (\neg A \wedge B))$

**De Morgan's Laws**

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

19-c

## A Simple Decision Procedure for Propositional Logic

### An Algorithm for Satisfiability

To check whether  $\alpha$  is satisfiable, form the truth table for  $\alpha$ . If there is a row in which **1** appears as the value for  $\alpha$ , then  $\alpha$  is satisfiable. Otherwise,  $\alpha$  is unsatisfiable.

### An Algorithm for Tautological Implication

To check whether  $\{\alpha_1, \dots, \alpha_k\} \models \beta$ , check the satisfiability of  $(\alpha_1 \wedge \dots \wedge \alpha_k) \wedge (\neg\beta)$ . If it is unsatisfiable, then  $\{\alpha_1, \dots, \alpha_k\} \models \beta$ , otherwise  $\{\alpha_1, \dots, \alpha_k\} \not\models \beta$ .

What is the complexity of this algorithm?

18-b

## More Tautologies

**Implication**

- $(A \rightarrow B) \leftrightarrow (\neg A \vee B)$

**Excluded Middle**

- $A \vee \neg A$

**Contradiction**

- $\neg(A \wedge \neg A)$

**Contraposition**

- $(A \rightarrow B) \leftrightarrow (\neg B \rightarrow \neg A)$

**Exportation**

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

20-d

## First-Order Logic: Motivation

Propositional logic is not powerful enough for many applications.

For example, propositional logic cannot reason about natural numbers directly.

In general, to reason about infinite domains or to express properties which are more abstract, a more expressive logic is required.

*First-order logic* is the most common logic of choice for handling tasks that require more power than that offered by propositional logic.

21

## First-Order Logic: Syntax

### Abbreviations

- Other propositional connectives:  $\vee, \wedge, \leftrightarrow$
- Existential quantifier:  $\exists x p(x) \leftrightarrow \neg \forall x \neg p(x)$

Each predicate and function symbol has an associated *arity*, a natural number indicating how many arguments it takes.

Equality is a special predicate symbol of arity 2.

Constant symbols can also be thought of as functions of arity 0.

A *first-order language* must first specify its parameters.

23

## First-Order Logic: Syntax

As with propositional logic, expressions in first-order logic are made up of sequences of symbols.

Symbols are divided into *logical symbols* and *non-logical symbols* or *parameters*.

### Logical Symbols

- Parentheses:  $(, )$
- Propositional connectives:  $\rightarrow, \neg$
- Variables:  $v_1, v_2, \dots$
- Universal quantifier:  $\forall$

### Parameters

- Equality symbol (optional):  $=$
- Predicate symbols: e.g.  $p(x), x > y$
- Constant symbols: e.g.  $0, John, \pi$
- Function symbols: e.g.  $f(x), x + y, x +_{[2]} y$

22

## First-Order Logic: Terms

The first important concept on the way to defining well-formed formulas is that of *terms*.

For each function symbol  $f$  of arity  $n$ , we define a term-building operation  $\mathcal{F}_f$ :

$$\mathcal{F}_f(\alpha_1, \dots, \alpha_n) = f\alpha_1, \dots, \alpha_n$$

Note that we are using prefix notation to avoid ambiguity.

The set of *terms* is the set of expressions generated from the constant symbols and variables by the  $\mathcal{F}_f$  operations.

Terms are expressions which name objects.

### Theorem

The set of terms is freely generated from the set of variables and constant symbols by the  $\mathcal{F}_f$  operations.

24

## First-Order Logic: Formulas

### Atomic Formulas

An *atomic formula* is an expression of the form:  $Pt_1, \dots, t_n$  where  $P$  is a predicate symbol of arity  $n$  and  $t_1, \dots, t_n$  are terms.

If the language includes the equality symbol, we consider the equality symbol as a predicate of arity 2.

### Formulas

We define the following formula-building operations:

- $\mathcal{E}_{\neg}(\alpha) = (\neg\alpha)$
- $\mathcal{E}_{\rightarrow}(\alpha, \beta) = (\alpha \rightarrow \beta)$
- $\mathcal{Q}_i(\alpha) = \forall v_i \alpha$

The set of *well-formed formulas* (*wffs* or just *formulas*) is the set of expressions generated from the atomic formulas by the operations  $\mathcal{E}_{\neg}$ ,  $\mathcal{E}_{\rightarrow}$ , and  $\mathcal{Q}_i$   $i = 1, 2, \dots$

This set is also freely generated.

25

## First-Order Logic: Semantics

In propositional logic, the truth of a formula was determined by a *truth assignment* over the propositional variables.

In first-order logic, we use a *model* (also known as a *structure*) to determine the truth of a formula.

A *signature* is a set of non-logical symbols (predicates, constants, and functions). Given a signature  $\Sigma$ , a model  $M$  of  $\Sigma$  consists of the following:

1. A nonempty set called the *domain* of  $M$ , written  $\text{dom}(M)$ . Elements of the domain are called elements of the model  $M$ .
2. A mapping from each constant  $c$  in  $\Sigma$  to an element  $c^M$  of  $M$ .
3. A mapping from each  $n$ -ary function symbol  $f$  in  $\Sigma$  to  $f^M$ , an  $n$ -ary function from  $[\text{dom}(M)]^n$  to  $\text{dom}(M)$ .
4. A mapping from each  $n$ -ary predicate symbol  $p$  in  $\Sigma$  to  $p^M \subseteq [\text{dom}(M)]^n$ , an  $n$ -ary relation on the set  $\text{dom}(M)$ .

27

## Free and Bound Variables

We define by recursion what it means for a variable  $x$  to *occur free* in a *wff*  $\alpha$ :

- If  $\alpha$  is an atomic formula, then  $x$  occurs free in  $\alpha$  iff  $x$  occurs in  $\alpha$ .
- $x$  occurs free in  $(\neg\alpha)$  iff  $x$  occurs free in  $\alpha$ .
- $x$  occurs free in  $(\alpha \rightarrow \beta)$  iff  $x$  occurs free in  $\alpha$  or in  $\beta$ .
- $x$  occurs free in  $\forall v_i \alpha$  iff  $x$  occurs free in  $\alpha$  and  $x \neq v_i$ .

If  $\forall v_i$  appears in  $\alpha$ , then  $v_i$  is said to be *bound* in  $\alpha$ .

Note that a variable can both occur free and be bound in  $\alpha$ . Because this can be confusing, we typically require the set of free and bound variables to be disjoint.

If no variable occurs free in a *wff*  $\alpha$ , then  $\alpha$  is a *sentence*.

26

## First-Order Logic: Semantics

### Example

Consider the signature corresponding to the language of set theory which has a single predicate symbol  $\in$  and a single constant symbol  $\emptyset$ .

A possible model  $M$  for this signature has  $\text{dom}(M) = \mathcal{N}$ , the set of natural numbers,  $\in^M = <$ , and  $\emptyset^M = 0$ .

Now consider the sentence  $\exists x \forall y \neg y \in x$ .

The translation of the sentence in the model  $M$  is that there is a natural number  $x$  such that no other natural number is smaller than  $x$ .

Since  $0$  has this property, the sentence is true in this model.

We will often use a shorthand when discussing both signatures and models. The signature shorthand lists each symbol in the signature.

The model shorthand lists the domain and the interpretation of each symbol of the signature.

The signature for set theory can thus be described as  $(\in, \emptyset)$ , and the above model as  $(\mathcal{N}, <, 0)$ .

28

## First-Order Logic: Semantics

Given a model  $M$ , a *variable assignment*  $s$  is a function which assigns to each variable an element of  $M$ .

Given a wff  $\phi$ , we say that  $M$  *satisfies*  $\phi$  with  $s$  and write  $\models_M \phi[s]$  if  $\phi$  is true in the model  $M$  with variable assignment  $s$ .

To define this formally, we first define the extension  $\bar{s} : T \rightarrow \text{dom}(M)$ , a function from the set  $T$  of all terms into the domain of  $M$ :

1. For each variable  $x$ ,  $\bar{s}(x) = s(x)$ .
2. For each constant symbol  $c$ ,  $\bar{s}(c) = c^M$ .
3. If  $t_1, \dots, t_n$  are terms and  $f$  is an  $n$ -ary function symbol, then  $\bar{s}(ft_1, \dots, t_n) = f^M(\bar{s}(t_1), \dots, \bar{s}(t_n))$ .

The existence of a unique such extension  $\bar{s}$  follows from the recursion theorem and the fact that the terms are freely generated.

Note that  $\bar{s}$  depends on both  $s$  and  $M$ .

29

## Logical Definitions

Suppose  $\Sigma$  is a signature. A  $\Sigma$ -*formula* is a well-formed formula whose non-logical symbols are contained in  $\Sigma$ .

Let  $\Gamma$  be a set of  $\Sigma$ -formulas. We write  $\models_M \Gamma[s]$  to signify that  $\models_M \phi[s]$  for every  $\phi \in \Gamma$ .

If  $\Gamma$  is a set of  $\Sigma$ -formulas and  $\phi$  is a  $\Sigma$ -formula, then  $\Gamma$  *logically implies*  $\phi$ , written  $\Gamma \models \phi$ , iff for every model  $M$  of  $\Sigma$  and every variable assignment  $s$ , if  $\models_M \Gamma[s]$  then  $\models_M \phi[s]$ .

We write  $\psi \models \phi$  as an abbreviation for  $\{\psi\} \models \phi$ .

$\psi$  and  $\phi$  are *logically equivalent* iff  $\psi \models \phi$  and  $\phi \models \psi$ .

A  $\Sigma$ -formula  $\phi$  is *valid*, written  $\models \phi$  iff  $\emptyset \models \phi$  (i.e.  $\models_M \phi[s]$  for every  $M$  and  $s$ ).

31

## First-Order Logic: Semantics

### Atomic Formulas

1.  $\models_M = t_1 t_2[s]$  iff  $\bar{s}(t_1) = \bar{s}(t_2)$ .
2. For an  $n$ -ary predicate symbol  $P$ ,  $\models_M P t_1, \dots, t_n[s]$  iff  $\langle \bar{s}(t_1), \dots, \bar{s}(t_n) \rangle \in P^M$ .

### Other wffs

1.  $\models_M (\neg \phi)[s]$  iff  $\not\models_M \phi[s]$ .
2.  $\models_M (\phi \rightarrow \psi)[s]$  iff  $\not\models_M \phi[s]$  or  $\models_M \psi[s]$ .
3.  $\models_M \forall x \phi[s]$  iff  $\models_M \phi[s(x|d)]$  for every  $d \in \text{dom}(M)$ .

$s(x|d)$  signifies the function which is the same as  $s$  everywhere except at  $x$  where its value is  $d$ .

Again, the well-formedness of this definition depends on the recursion theorem and the fact that *wffs* are freely generated.

30

## Computability

The important notion of *computability* relies on a formal model of computation.

Many formal models have been proposed:

1. General recursive functions defined by means of an equation calculus (Gödel-Herbrand-Kleene)
2.  $\lambda$ -definable functions (Church)
3.  $\mu$ -recursive functions and partial recursive functions (Gödel-Kleene)
4. Functions computable by finite machines known as Turing machines (Turing)
5. Functions defined from canonical deduction systems (Post)
6. Functions given by certain algorithms over a finite alphabet (Markov)
7. Universal Register Machine-computable functions (Shepherdson-Sturgis)

### Fundamental Result

All of these (and many other) models of computation are equivalent. That is, they give rise to the same class of functions.

32

## Computability and Decidability

All of these models are equivalent to what can be achieved by a computer with any standard programming language, given arbitrary (but finite) time and memory.

### Church's Thesis

A notion known as Church's thesis states that all models of computation are either equivalent to or less powerful than those just described.

We will accept Church's thesis and thus define a function to be *computable* if we can describe precisely (using any model of computation) how to compute it. Such a description will be called an *effective procedure*.

### Decidability

Given a universal set  $U$ , a set  $S \subseteq U$  is decidable if there exists a computable function  $f : U \rightarrow \{0, 1\}$  such that  $f(x) = 1$  iff  $x \in S$ .

33-a

## Semi-Decidability

Suppose we wish to determine whether  $\Sigma \models \alpha$  where  $\Sigma$  is infinite. In general, this is not decidable.

But we can obtain a weaker result:

A set  $A$  is *semi-decidable* (or *effectively enumerable*) if there is an effective procedure which lists, in some order, every member of  $A$ .

Note that if  $A$  is infinite, then the procedure will never finish, but every member of  $A$  must appear in the list after some finite amount of time.

### Theorem

A set  $A$  of expressions is effectively enumerable iff there is an effective procedure which, given any expression  $\alpha$ , produces the answer "yes" iff  $\alpha \in A$ .

35

## Decidability

### Some decidable sets

- For a given finite set of wff's  $\Sigma$ , the set of all *tautological consequences* of  $\Sigma$  (i.e.  $\{\alpha \mid \Sigma \models \alpha\}$ ) is decidable.

The truth table algorithm given earlier decides  $\Sigma \models \alpha$ .

- The set of tautologies is decidable.

The set of tautologies is just the set of tautological consequences of the empty set.

### Existence of undecidable sets

A simple argument shows the existence of undecidable sets of expressions: an algorithm is completely determined by its finite description. Thus, there are only countably many effective procedures. But there are uncountably many sets of expressions.

Why?

34-d

## Semi-Decidability

### Theorem

A set is decidable iff both it and its complement (with respect to a given universal set) are effectively enumerable.

### Proof

Alternate between running the procedure for the set and the procedure for its complement. One of them will eventually produce "yes".

### Properties of decidable and semi-decidable sets

Decidable sets are closed under union, intersection, and complement.

Semi-decidable sets are closed under union and intersection.

36-b

## Proofs and Deduction

A *proof* is a finite sequence of fixed indisputable steps.

A proof is built from *axioms*, facts which we accept without proof, and *theorems*, which are facts derived from axioms using an agreed-upon set of *rules of inference*.

Because the proof is finite and each step conforms to a predetermined set of rules, the question of whether a given sequence of steps is a proof is decidable.

There are many possible choices for the axioms and rules of inference.

A particular choice of axioms and rules of inference is often referred to as a *calculus*.

Rules of inference are often written in the following format with the given formulas above and the deduced formula below:

$$\frac{\alpha, \alpha \rightarrow \beta}{\beta}.$$

37

## Soundness and Completeness

An important question for any calculus is its relationship to the semantic notion of validity.

If only valid formulas are deducible, then the calculus is said to be *sound*.

If all valid formulas are deducible, then the calculus is said to be *complete*.

The existence of a sound and complete calculus for first-order logic is a important result which demonstrates that it is a reasonable model of mathematical thinking.

Unfortunately, for most languages, the set of valid formulas is *not* decidable. In fact, as long as the language contains at least one two-place predicate symbol, the set of valid formulas is undecidable.

39

## Proofs and Deduction

A *deduction of  $\phi$  from  $\Gamma$*  is a sequence  $\langle \alpha_0, \dots, \alpha_n \rangle$  of formulas such that  $\alpha_n = \phi$  and for each  $i \leq n$  either

- $\alpha_i$  is in  $\Gamma \cup \Lambda$ , where  $\Lambda$  is the set of axioms, or
- $\alpha_i$  is obtainable from  $\alpha_{j_1}, \dots, \alpha_{j_n}$  using a rule of inference, where each  $j_k < i$ .

If such a deduction exists, we say that  $\phi$  is *deducible* from  $\Gamma$  or that  $\phi$  is a *theorem* of  $\Gamma$ , and we write  $\Gamma \vdash \phi$ .

Note that although this is a well-defined inductive definition, the set is typically not freely generated, so there may be more than one deduction of the same formula.

38

## Theories

A *theory* is a set of first-order sentences *closed under logical implication*.

Thus,  $T$  is a theory iff  $T$  is a set of sentences and if  $T \models \sigma$ , then  $\sigma \in T$  for every sentence  $\sigma$ .

*Examples*

- For a given signature, the smallest possible theory consists of exactly the valid sentences over that signature.
- The largest theory for a given signature is the set of all sentences. It is the only unsatisfiable theory. Why?

40

## Theories

For a class  $\mathcal{K}$  of models over a given signature  $\Sigma$ , define the *theory of  $\mathcal{K}$*  as

$$\text{Th}\mathcal{K} = \{\sigma \mid \sigma \text{ is a } \Sigma\text{-sentence which true in every model in } \mathcal{K}\}.$$

Suppose  $\Gamma$  is a set of sentences. Define the set  $\text{Cn } \Gamma$  of *consequences* of  $\Gamma$  to be  $\{\sigma \mid \Gamma \models \sigma\}$ .

A theory  $T$  is *complete* iff for every sentence  $\sigma$ , either  $\sigma \in T$  or  $(\neg\sigma) \in T$ .

A theory  $T$  is *axiomatizable* iff there is a decidable set  $\Gamma$  of sentences such that  $T = \text{Cn } \Gamma$ .

A theory  $T$  is *finitely axiomatizable* iff  $T = \text{Cn } \Gamma$  for some finite set  $\Gamma$  of sentences.

## Validity and Satisfiability Modulo Theories

Given a  $\Sigma$ -theory  $T$ , a  $\Sigma$ -formula  $\phi$  is

1. *T-valid* if  $\models_M \phi[s]$  for all models  $M$  of  $T$  and all variable assignments  $s$ .
2. *T-satisfiable* if there exists some model  $M$  of  $T$  and variable assignment  $s$  such that  $\models_M \phi[s]$ .
3. *T-unsatisfiable* if  $\not\models_M \phi[s]$  for all models  $M$  of  $T$  and all variable assignments  $s$ .

The *validity problem* for  $T$  is the problem of deciding, for each  $\Sigma$ -formula  $\phi$ , whether  $\phi$  is  $T$ -valid.

The *satisfiability problem* for  $T$  is the problem of deciding, for each  $\Sigma$ -formula  $\phi$ , whether  $\phi$  is  $T$ -satisfiable.

Similarly, one can define the *quantifier-free validity problem* and the *quantifier-free satisfiability problem* for a  $\Sigma$ -theory  $T$  by restricting the formula  $\phi$  to be quantifier-free.