Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Topic 23
Mathematical Logic

- The **prerequisite** for following this part of the lecture is that you understand the mathematical concepts of sets, functions and algebras as covered in earlier lectures.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- The **aims** are to cover the concepts of Boolean algebra, propositional and predicate logic, to cover the concepts of proof theory and model theory and to cover the concept of axiom systems and exemplify its application in abstract specifications.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The **objective** is to help ensure that the course student becomes reasonably fluent in the use of logic as a specification tool and to begin the long road in ensuring that the course student will eventually become reasonably versatile in logic reasoning.
- The **treatment** is semiformal to fully formal.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Our treatment of Logic will spread over several lecture parts:
  1. A Language of Boolean Ground Terms
  2. Languages of Propositional Logic
  3. Languages of Predicate Logic
  4. Axiom Systems

Mathematical logic is, without any doubt, the most important mathematical subdiscipline of software engineering.

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Characterisation 9.55 By a *mathematical logic* we mean a formal language:

- A *syntax* defining an infinite set of formulas,
- and a "*semantics*" — here in the form of
  - ⋆ a set of *axioms* concerning these formulas
  - ⋆ and a set of *rules of inference* over these formulas

. ■

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Logic is the study of reasoning.
- Logic was, for a long time, part of philosophy.
- Mathematical logic is the study of the kind of reasoning done by mathematicians, and
- mathematical logic was, for some time, a stepchild of mathematics.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- We shall basically be using mathematical logic
  - ⋆ as undoubtedly the most important part of our specification notation.
- That is,
  - ⋆ we shall be using all the sublanguages of mathematical logic:
    - ◇ the sublanguage of Boolean ground terms,
    - ◇ the sublanguage of propositions and
    - ◇ the sublanguage of predicates.
- Therefore it is important that the course student — from the very beginning, that is, now! — is at ease with many of the concepts of mathematical logic.
- This, then, is the purpose of these next lectures:
  - ⋆ to teach you those concepts, and
  - ⋆ to teach you how to express yourself, formally, in those sublanguages.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- Correctness of software, and proving properties of their specifications and implementations, are concerns of core importance.
- The languages (i.e., tools) and techniques of mathematical logic are used in securing fulfillment of desired properties.
- We shall be covering, also, some of the proof aspects of mathematical logic.
- But our presentation in this lecture is from the point of view of mathematical logic as an abstract specification language.
- We will not cover theories of mathematical logic, but refer to many good textbooks.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Topic 24
The Issues

- We shall first treat basically nine issues of logics, including three sublanguages:
  - ⋆ (i) a language of Boolean-valued ground terms,
  - ⋆ (ii) a language of Boolean-valued propositional expressions, and
  - ⋆ (iii) a language of Boolean-valued predicate expressions.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- And we shall also cover some diverse issues:
  - ⋆ (iv) Boolean-valued expressions,
  - ⋆ (v) **chaos** — undefined expressions,
  - ⋆ (vi) axiom and inference systems,
  - ⋆ (vii) proof systems,
  - ⋆ (viii) the axioms of the logic languages, and the axiom definition facility of RSL, and
  - ⋆ (ix) the meaning of the **if ... then ... else ... end** clause.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- We first survey these nine issues, then we treat the three languages in more detail.
- But, before that, we survey the distinction between *proof-theoretic* and *model-theoretic* logic. That distinction will bring out the distinctions between syntax and semantics, between provable and true, and between completeness and soundness.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Language of Boolean Ground Terms

- First, there is the *Boolean ground term algebra*, or simply the *Boolean calculus*, its syntax, semantics and pragmatics.
- We refer to the Boolean ground term algebra by the type name **Bool.**

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

- Syntactically the *Boolean algebra*
  - ⋆ is a *language of ground terms*,
  - ⋆ having a syntax including:
    - ◇ Boolean (constant) literals (**true** and **false**),
    - ◇ a set of connectives: $\{\sim, \wedge, \vee, \Rightarrow, =, \neq, \equiv\}$,
    - ◇ a set of (syntax) rules for forming ground terms,
  - ⋆ and a set of axioms relating ground terms and connectives, a calculus.

true, **false**, $\sim$**true**, $\sim$**false**, **true** $\wedge$ **false**, $\sim$**true** $\wedge$ **false**, ...

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- Semantically we have
  - ⋆ truth tables for these connectives, and
  - ⋆ interpretation rules.
- Speaking on the pragmatics of the Boolean ground term algebra, with this (ground term) algebra there is little we can express.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Language of Propositional Expressions

- Next, we present the *propositional calculus*, its syntax, semantics and pragmatics.
- The propositional calculus builds on the language of Boolean ground terms.
- There is the syntax of propositional (operator/operand) expressions built from Boolean literals, connectives, and variable identifiers, axioms and inference rules.
- The axioms and inference rules define the calculus part of the propositional calculus.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- Variables are intended, in the semantics, to denote truth values.

  true, **false**, ∼**true**, ∼**false**, **true** ∧ **false**, ∼**true** ∧ **false**, ...
  a, b, ..., a ∧ **true**, a ∧ b, ...

- There are the semantics rules (an evaluation procedure) for interpreting propositional expressions.
- And there is the pragmatics: With the propositional calculus we can express a few more things than with just Boolean ground terms.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Language of Predicate Expressions

- Finally, we have the *predicate calculus*, with its syntax, semantics and pragmatics.
- The predicate calculus includes the propositional calculus.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Thus there is the syntax of predicate (operator/operand) expressions,
  - ⋆ including propositional expressions,
    - ◇ extended with constant values of any type,
    - ◇ variables denoting such values, and hence
    - ◇ operator/operand expressions also over these,
  - ⋆ as well as quantified expressions ($\forall, \exists$),
  - ⋆ and axioms and inference rules.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

⋆ The axioms and inference rules define the calculus part of the predicate calculus.

true, **false**, ∼**true**, ∼**false**, **true** ∧ **false**, ∼**true** ∧ **false**, ...
a, b, ..., a ∧ **true**, a ∧ b, ...
∀ x:X · **true**, ∀ x:X · x ∧ ..., ∃ x:X · x ∧ ...

- There are the semantics rules for interpreting (evaluating) predicate expressions — leading to truth values (or **chaos**!).
- And there is the pragmatics: With the predicate calculus we can express quite a lot. It is sufficient for a long while!

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Boolean-Valued Expressions

Syntactically we can thus speak of four categories of expressions: Boolean ground terms, propositional expressions, predicate expressions and quantified expressions.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Figure 9.12 informally indicates

- that *Boolean ground term expressions* syntactically are a proper subcategory of *propositional expressions;*
- that *propositional expressions* syntactically are a proper subcategory of *predicate expressions;*
- that *quantified expressions* syntactically are a proper subcategory of *predicate expressions;* but
- that *quantified expressions* syntactically are not a proper subcategory of *propositional expressions.*

It also expresses that all are *Boolean-valued expressions.*

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

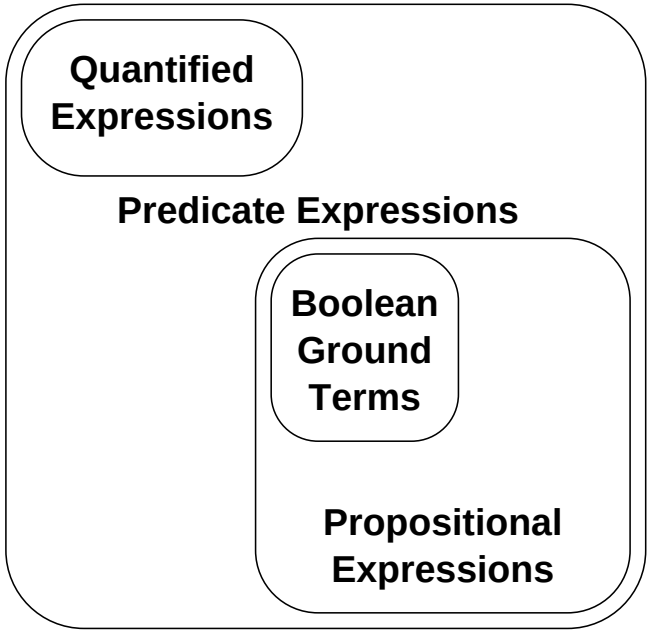

Figure 9.12: Languages of Boolean-valued expressions

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

"chaos" — Undefined Expression Evaluations

- We reintroduce, at this point, the literal **chaos**.
  - ⋆ It pertains to possible evaluations (i.e., of finding the values) of arbitrary expressions.
  - ⋆ If an expression cannot be evaluated ($e/0$ never evaluates!), then its value is said to be **chaos**.
  - ⋆ That is, we can speak of never terminating, or undefined evaluations, and we give the name **chaos** to the "value", i.e., the result of such evaluations.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Axiom Systems and Inference Rules

- Just as we have the calculus of integers, that is,
  - ⋆ rules for adding, subtracting, multiplying and integer-dividing integers,
  - ⋆ and rules for eliminating certain additions, subtractions, multiplications and divisions:

$$0 + a = a,\ 1 \times a = a,\ 0 \times a = 0,$$
$$a/1 = a,\ 0/a = 0 \text{ (where } a \neq 0),\ \text{ etc.}$$

  - ⋆ so we have rules, in general called inference rules,
  - ⋆ for "reducing" or "rewriting" syntactic logic expressions into other (usually simpler) such expressions.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

- Axioms and inference rules (of some logic) together make up the calculus for that logic.
- A logic is defined by its axioms and inference rules.
- We shall, in subsequent sections introduce, various axiom systems.

Axioms and Axiom Systems

- An *axiom*
  - ⋆ is a predicate expression with free variables.
  - ⋆ These variables designate arbitrary predicate expressions.
  - ⋆ An axiom thus designate an infinity of predicates without variables, where all (former free) variables have been replaced by propositions.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- ⋆ A "classical" logic axiom is:

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

  - ⋆ ϕ is the free variable.
  - ⋆ It reads: Either ϕ holds, or ϕ does not hold.
  - ⋆ The axiom is called the axiom of the *excluded middle*,
  - ⋆ The pragmatics of an axiom, of a logic, is that it represents, in some or all semantics of that logic, a self-evident truth.

- An *axiom system*
  - ⋆ is a collections of one or more axioms.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Inference Rules

- An *inference rule*
  - ⋆ is a pair:
    - ◇ a set of predicates with free variables (the premise),
    - ◇ and an inferred predicate with some of the same free variables (the conclusion).
  - ⋆ The most famous logic inference rule is that of *modus ponens*:

$$\frac{P, P \supset Q}{Q}$$

  - ⋆ P and Q are the free variables.
  - ⋆ It reads:
    - ◇ If we know that P holds
    - ◇ and that $P \supset Q$ holds,
    - ◇ then we can infer (conclude) that Q holds.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The pragmatics of an inference rule, of a logic, is that it represents, in some or all semantics of that logic,
  - ⋆ a self-evident way of reasoning,
  - ⋆ from one set of logic expressions
  - ⋆ to the next, or to another logic expression.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Proof Systems

- By a proof system for a logic language we mean:
  - ⋆ a set of axiom schemes,
  - ⋆ a set of rules of inference,
  - ⋆ and a set of theorems provable from the axiom schemes and rules of inference.
- The latter can be considered as being axioms.
- Some theorems may be reformulated as "additional" rules of inference:[14]

$$\frac{\Gamma, \phi \vdash \psi}{\xi}$$

The verifier, a person or a mechanised system, has "more to choose from"!

[14] $\frac{\Gamma, \phi \vdash \psi}{\xi}$ reads: If, assuming the set of axioms (etc.) Γ, ψ can be proved from ϕ, then ξ holds (i.e., has [thus] been proved).

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

In our presentation of proof systems, in particular that of RSL, we present

- not only *not* the entire proof systems,
- but also *not* the full details of how to carry out full proofs,
- and certainly *not* how to do even small proofs using available theorem prover or proof assistant software systems.

To learn how to do real proofs for real developments is a deep study by itself.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

- Summarising we can say:
  - ⋆ Proof systems are specially tailored versions of
    - ◇ axiom schemes
    - ◇ and rules of inferences —
    - ◇ augmented by theorems and
    - ◇ special syntactic conventions on how to present proofs.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

A Note on Two Axiom Systems

- Axioms are self-evident truths, i.e., can be considered laws.
- But we have to keep track of two kinds of notions of axioms and axiom systems:
  - ⋆ The axioms that define proof systems of logic languages, including RSL,
  - ⋆ and the axioms that a user of RSL defines when specifying properties of sorts and functions.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- The two relate as follows:
  - ⋆ The axioms of the proof systems of logic languages, like RSL, are given, a priori, and are not expressed in those same languages. However, the student may get the impression that RSL's proof system is defined in RSL, since the axioms look very much like the axiom definition facility of RSL.
  - ⋆ The axioms that are expressed in RSL, using RSL Boolean valued and other expressions, and which rely on RSL's proof system when proving properties of what these user-defined axioms express.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- In the next lecture parts on the logic languages of Boolean ground terms, propositions and predicates, we shall be speaking about the axioms of RSL's proof system.
- Later we shall, in contrast, illustrate the use of RSL' axiom definition facility in defining data types like *Euclid's plane geometry*, *natural number (Peano's axiom system)*, *simple sets*, and *simple lists*.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

The "if ... then ... else ... end" Connective

- The **if**...**then**...**else**...**end** construct "anchors" around a basic understanding of logic.
- We therefore explain this construct.
- Let e be:

if b **then** e′ **else** e″ **end**

- e is a syntactic construct of, for example, RSL. It allows b to evaluate to a value of any type and to **chaos** (which has no type).

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The expression **e** only makes sense if **b** evaluates to **false** or **true**:

if false then e′ **else** e″ **end** ≡ e″
if true then e′ **else** e″ **end** ≡ e′
if chaos then e′ **else** e″ **end** ≡ **chaos**

- If **b** evaluates to any other value **chaos** is still the result.
- **chaos** stands for *chaotic behaviour* of the result of evaluating an expression, including nontermination.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- *Nonstrictness* of a functional, like the *distributed fix*, **if**...**then**...**else**...**end**, means that applying the functional to arguments that may evaluate to **chaos** does not necessarily lead to **chaos**:

if true then e′ **else chaos end** ≡ e′
if false then chaos else e″ **end** ≡ e″

We refer to **if** ...**then** ...**else** ...**end** as a *distributed-* or *mix-fix connective.*

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- We are building up our treatment of logics in small, easy steps.
- In this section we have basically identified three languages of logic,
  - ⋆ a language of Boolean ground terms,
  - ⋆ a language of propositions and
  - ⋆ a language of predicates.
- Each of these languages will be dealt with in more detail later.
- But first we treat a number of issues common to the three languages.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Topic 25
Proof Theory Versus Model Theory

- Earlier we have made the distinction between the syntax and the semantics of a language.
- Now we will elucidate this distinction.
- In this lecture we shall assume a classical two-valued logic.

Syntax

- What we write is syntax.
- When we manipulate written text, in some language, using certain (for example inference) rules and axioms, and thereby obtain other text in the same language, then these rules are basically of syntactic nature.

Example 9.35 *Differentiation of Analytic Expressions, I:*

- We take, as an example, that of the formal language of analytic expressions
- And we take as rules those which define differentiation

Analytic Expression	Rule of Differentiation
$y : a$	$\frac{\partial y}{\partial x} = \frac{\partial a}{\partial x} \rightsquigarrow 0$
$y : x$	$\frac{\partial y}{\partial x} = \frac{\partial x}{\partial x} \rightsquigarrow 1$
$y : x^n$	$\frac{\partial y}{\partial x} = \frac{\partial (x^n)}{\partial x} \rightsquigarrow n \times x^{n-1}$
$y : f(x) + g(x)$	$\frac{\partial y}{\partial x} = \frac{\partial (f(x)+g(x))}{\partial x} \rightsquigarrow \frac{\partial (f(x))}{\partial x} + \frac{\partial (g(x))}{\partial x}$
$y : f(x) \times g(x)$	$\frac{\partial y}{\partial x} = \frac{\partial (f(x)\times g(x))}{\partial x} \rightsquigarrow \frac{\partial (f(x))}{\partial x} \times g(x) + \frac{\partial (g(x))}{\partial x} \times f(x)$
etc.	*etc.*

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- We observe that the rules of differentiation
  - ⋆ when applied to any analytic expression
  - ⋆ terminate
  - ⋆ with the result being an analytic expression.
- In other words, the language plus the rules remain syntactic.

■

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- The notions of proofs and theorems (in logic) are syntactic notions.
- There is a large body of theory that deals only with the syntax of any, or some, logic language(s).
- Similarly, there is a large body of theory that deals only with the differentiability of analytic expressions, also a syntactic theory.
- Mathematical logic can be pursued, at length and in depth, while remaining at the syntactic level.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Semantics

- What we mean by the written text, in contrast, is semantics.

Example 9.36 *Differentiation of Analytic Expressions, II:*

- ⋆ Why we perform differentiation is of no concern to the rules of differentiation as they are being applied.
- ⋆ The semantics of an analytic expression may express distance covered over time.
- ⋆ Differentiation wrt. time may therefore be done in order to express the velocity.
- ⋆ Differentiation wrt. time performed twice may therefore be done in order to express the acceleration.

■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- Semantics is about truth, about the 'holding' or 'not holding' of a logical sentence.
- Thus the Boolean ground terms **false** and **true** denote the semantic values *falsity* and *truth*, respectively.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.37 *Meaning of Logical Expressions:*

- A logical expression, ϕ, may mean that it designates the properties of a requirements prescription.
- Another logical expression, ψ, may mean that it designates the properties of a software specification.
- The logical expression, $\psi \supset \phi$, may then mean that the software specification implements the requirements.

■

Syntax Versus Semantics

- To sum up:
- When speaking in the syntactic realm of a logic language the logic expressions are mere symbols — we are not interested in their meaning. We manipulate strings of symbols using the axioms and rules of inference.
- When speaking in the semantic realm of a logic language the logic expressions denote values, and these values are obtained through interpretation.
  - ⋆ There is a *context* which, among others, maps expression symbols (including variable identifiers) to their truth values.
  - ⋆ Different contexts (we say different 'worlds') may map the same variable identifier to different truth values.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Formal Logics: Syntax and Semantics

- The various logic languages, their syntax and semantics, all manifest formal systems.
- A formal logic system, syntactically, consists of several parts.
  - ⋆ First, it contains (i) a logic language given by some concrete grammar which elucidates
    - ◇ constant and function (i.e., operation) literals, for example, **false, true, chaos,** ¬ (or ∼), ∧, ∨, and ⊃,
    - ◇ variable, function and predicate identifiers,
    - ◇ delimiters (like commas: ",", parentheses: "(", ")", etc.),
    - ◇ and their combination (say in terms of a set of `BNF` rules).

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Second, a formal logic system, syntactically, also consists of
  - ⋆ (ii) an axiom system:
    - ◇ a set of axioms, viz.:

$$\phi \vee \neg\phi.$$

    - ◇ In other words, the axiom system is a subset of sentences of the language,
    - ◇ in which variable identifiers (ϕ) are metalinguistic: they designate proper sentences (viz.: $(P \vee Q) \wedge R$) of the language.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Finally, a formal logic system, syntactically, also consists of
  - ⋆ (iii) a set of rules of inference:
    - ◇ a set of pairs of antecedents and consequents, viz.:

$$\frac{\phi, \phi \supset \psi}{\psi}$$

    - ◇ The former is a set of sentences, and
    - ◇ the latter is a sentence,
    - ◇ such that all variable identifiers of these sentences are metalinguistic.
    - ◇ They designate proper sentences of the language.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

More on the Semantics of Formal Logic Systems

- Semantically, a formal system extends its syntax along two lines.
- Along one line, a context is provided, something which to every symbol of the language associates appropriate semantic notions.
  - ⋆ To literals (**false, true, chaos**) one associates the semantic truth values (`ff, tt` or *falsity*, *truth*), respectively the semantic undefined value ($\bot$).
    - ◇ $\bot$ "propagates" by making any expression evaluation in which it occurs denote that value.
  - ⋆ To variable identifiers one associates some proper truth or other value,
  - ⋆ etcetera.

  What the "etcetera" stands for will be revealed later, suffice it here to hint at operator, function and predicate symbols.
- Along the other line, a semantics prescribes an evaluation (an interpretation) procedure which when applied to a sentence in a context results in a value: *falsity*, *truth* or $\bot$.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

More on the Syntax of Formal Logic Systems

- There are usually two parts to a formal system:
  - ⋆ One part, the *logical part* that is shared by all logic languages,
  - ⋆ and another, the *non-logical part*.
- The symbols that belong to the *logical part* are called the *logical symbols* of the system.
- The *connectives* are logical symbols:

$$\neg, \quad \vee, \quad \wedge, \quad \supset, \equiv$$

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- In the predicate calculi we additionally introduce:

$$f_1, \quad f_2, \quad \ldots, f_n, \quad \forall, \quad \exists$$

- where f_i are function symbols, and $\forall$ and $\exists$ are the universal, respectively the existential quantifiers.
- The non-logical symbols are given special interpretations:

$$+, \quad -, \quad \times, \quad /, \quad <, \quad \leq, \quad =, \quad >, \quad \geq, \quad \ldots$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The connectives are chosen to "mimic" every language use, with some more precision, of the terms:
  - ⋆ 'and' ($\wedge$),
  - ⋆ 'or' ($\vee$),
  - ⋆ 'not' ($\neg$),
  - ⋆ 'equal' ($\equiv$), and
  - ⋆ 'imply' ($\supset$).
- In $P \supset Q$
  - ⋆ P is called the *antecedent*.
  - ⋆ Q is called the *consequent*.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

On the Meaning of Material Implication, ⊃

- Let us dwell, for a moment, on the issue of the intended (semantic) meaning of implication ⊃:

$$P \supset Q$$

- When we say that a logical expression holds we mean that it evaluates to **true**.
- $P \supset Q$ reads:
  - ⋆ If P holds, then Q holds; if P then Q.

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.38 *Informal Uses of Implication, I:* Let us illustrate some examples of uses of implication.

- The deduction
  - ⋆ "the *jaberwocky* is a *tove;*
  - ⋆ all *toves* are *slithy;*
  - ⋆ therefore that *jaberwocky* is *slithy*",
- seems OK even though we have do not know what *jaberwocky*, *tove* and *slithy* means.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- What about
  - ⋆ "The air plane is a Boeing 737;
  - ⋆ therefore it has two engines"?
- That does not seem OK, even though its conclusion is true.
- It jumps to a conclusion that is not supported by the facts that are **explicitly** mentioned.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- What about:
  - ⋆ "the car is a Chrysler;
  - ⋆ therefore it has two engines"?
- We see this as palpable nonsense.
- We can repair the above
  - ⋆ "The air plane is a Boeing 737;
  - ⋆ all Boeing planes, except the 747, have two engines;
  - ⋆ therefore that plane has two engines."
- Now the reasoning is sound.
- And soundness does not depend on whether we understand the terms 'Boeing', 'engine', '737', or '747'.

■

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Following John Rushby we show an example,
- and then analyse possible semantics of the implication connective.

Example 9.39 *Informal Uses of Implication, II:*

- Consider the four implications:
  - ⋆ (1) $2+2=4 \supset$ Paris is the Capital of France;
  - ⋆ (2) $2+2=4 \supset$ London is the Capital of France;
  - ⋆ (3) $2+2=5 \supset$ Paris is the Capital of France; and
  - ⋆ (4) $2+2=5 \supset$ London is the Capital of France.
- What truth values can we ascribe to (1–4)?
  - ⋆ (1) is true because both the antecedent and the consequent are true.
  - ⋆ (2) is false because the consequent is false.
  - ⋆ (3) is what?
  - ⋆ (4) is what?
- To answer (3) and (4) we turn to the next analysis.

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Thus if, in $P \supset Q$, P does not hold, then we do not (based on what we have presented up till now) know whether Q holds,
- and hence we do not know whether $P \supset Q$ holds.
- If P holds, but Q does not hold, then our intuition dictates that $P \supset Q$ does not hold.
- So what are we to say about the holdings of $P \supset Q$ when P does not hold?
  - ⋆ If we say that $P \supset Q$ does not hold, when P and Q do not hold, then $P \supset Q$ is the same as $P \wedge Q$.
  - ⋆ If we say that $P \supset Q$ holds exactly when Q holds, then $P \supset Q$ is the same as Q.
  - ⋆ If we say that $P \supset Q$ holds exactly when Q does not hold, then $P \supset Q$ is the same as $P \equiv Q$.
  - ⋆ Thus we conclude that $P \supset Q$ holds when P and Q hold, and when P does not hold (irrespective of holding of Q).

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Metalinguistic Variables

- In axioms, such as:

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

- and in rules of inference, such as:

$$\frac{\phi, \phi \supset \psi}{\psi}$$

- the identifiers ϕ and ψ stand for arbitrary logic sentences.
- They are metalinguistic variables.
- In any particular use of logic in some specification we may have some propositions or some predicates P and Q.
- They can now be substituted in lieu of ϕ and ψ

$$P \vee \neg P$$

- respectively

$$\frac{P, P \supset Q}{Q}$$

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Since any Ps and Qs are acceptable we see that axiom and rules of inference really are schemes of axioms, respectively schemes of inference.
- That is, they stand for infinities of axioms and infinities of rules of inference.
- Given a metalinguistic variable, say ϕ, and given some instance of a propositional or predicate sentence, say P,
- we may express that P is to take the place of ϕ in some (designated) axiom scheme or in some (designated) rule of inference scheme as follows:

$$[\phi \mapsto P]$$

- The form $[\phi \mapsto P]$ is called a substitution specification clause.
- Substitution specifications may contain several clauses:

$$[\phi_1 \mapsto P_1, \phi_2 \mapsto P_2, \ldots, \phi_n \mapsto P_n]$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Issues Related to Proofs

Proofs

- Given a sentence ϕ. A *proof* of ϕ, from a set, Γ, of sentences
- is a finite sequence of sentences, $\phi_1, \phi_2, \ldots, \phi_n$,
- where $\phi = \phi_1$, where $\phi_n = \mathbf{true}$,
- and in which each ϕ_i is
  - ⋆ either an axiom,
  - ⋆ or a member of Γ,
  - ⋆ or follows from earlier ϕ_js by one of the rules of inference.
- We say that ϕ is *provable* from *assumptions* Γ,
- or simply Γ proves ϕ: $\Gamma \vdash \phi$
- Proofs and provability are syntactic notions, i.e., are notions of proof theory.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Theorems and Formal System Theories

- A *theorem* is a sentence that is provable without assumptions, that is purely from axioms and rules of inferences.
- We say that a *theory* of a given formal system is the set of all its theorems.
- Theorems and theories are syntactic notions, i.e., are notions of proof theory.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Consistency

- A formal system is *consistent* if it contains no sentence ϕ such that both ϕ and its negation $\neg\phi$ are theorems.
- It is a meta-theorem of all the two-valued logics that all sentences are provable in an inconsistent formal, two-valued logic system.
- Consistency is a syntactic notion, i.e., is a notion of proof theory.

Decidability

- A formal logic system is *decidable*
  - ⋆ if there is an algorithm which prescribes computations
  - ⋆ that can determine whether or not
  - ⋆ any given sentence in the system
  - ⋆ is a theorem (or not).

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Relating Proof Theory to Model Theory

- In modelling domains, requirements and software using logic, we are modelling some "worlds".
- So far we have emphasised the syntactic aspects of logic.
- To establish a relationship between the syntactic aspects of the sentences of a formal language and some world we must turn to semantics.
- The goal, then, of mathematical logic is to make sure that theorems are true in the chosen world, or worlds.
- We wish to make sure that the theorems we can prove will correspond to true statements about a chosen world, or all worlds.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Interpretation

- The connection between syntax and semantics is, as always, established through an interpretation, $\mathcal{I}$.
- So we start with a formal logic system, $\mathcal{L}$.
- An interpretation $\mathcal{I}$ identifies some chosen world, Ω,
- and associates a true or a false statements with each sentence of the formal system.
- Statements are of the kind: *"the logic expression ϕ (about such-and-such) is true in Ω"*, or *"the logic expression ϕ (about such-and-such) is false in Ω"*.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- The interpretation, $\mathcal{I}$, has two parts:
  - ⋆ A context, an environment, ρ, which to every symbol in $\mathcal{L}$, associates some value in Ω,
  - ⋆ and a procedure for evaluating any sentence ϕ in $\mathcal{L}$.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.40 *The Factorial and The List Reversal Functions:*

- Let ϕ be the sentence:

$$\exists F \bullet ((F(a) = b) \wedge \forall x \bullet (p(x) \supset (F(x) = g(x, F(f(x))))))$$

- which, model-theoretically, reads:
  - ⋆ there exists a mathematical function F
  - ⋆ such that (•) the following holds, namely:
  - ⋆ $F(a) = b$ (where a and b are not known, model-theoretically), and $\wedge$
  - ⋆ for every (i.e., all) x it is the case (•) that
  - ⋆ if $p(x)$ is true, then $F(x) = g(x, F(f(x)))$ is true
  - ⋆ (where x, g and f are not known, model-theoretically).

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Now there are (at least) two possible interpretations of ϕ.
- In the first interpretation we establish
  - ⋆ first the world Ω of natural numbers and operations on these,
  - ⋆ and then the specific context ρ:

$$
\begin{array}{l}
[\ \mathrm{F} \mapsto \mathrm{fact}, \\
\ \ \mathrm{a} \mapsto 1, \\
\ \ \mathrm{b} \mapsto 1, \\
\ \ \mathrm{f} \mapsto \lambda\ \mathrm{n.n}{-}1, \\
\ \ \mathrm{g} \mapsto \lambda\ \mathrm{m}.\lambda\ \mathrm{n.m}{+}\mathrm{n} \\
\ \ \mathrm{p} \mapsto \lambda\ \mathrm{m.m}{>}0\ ]
\end{array}
$$

- And we find that ϕ is true for the factorial function, fact.
- In other words, ϕ characterises properties of that function.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- In the second interpretation we establish
  - ⋆ first the world Ω of lists and operations on these:
  - ⋆ and then the specific context ρ:

$$
\begin{bmatrix}
\mathrm{F} \mapsto \mathrm{rev}, \\
\mathrm{a} \mapsto \langle\rangle, \\
\mathrm{b} \mapsto \langle\rangle, \\
\mathrm{f} \mapsto \mathbf{tl}, \\
\mathrm{g} \mapsto \lambda\ell_1.\lambda\ell_2.\ell_1\widehat{\ }\langle\mathbf{hd}\ \ell_1\rangle \\
\mathrm{p} \mapsto \lambda\ell.\ell{\neq}\langle\rangle
\end{bmatrix}
$$

- And we find that ϕ is true for the list reversal function, rev.
- In other words, ϕ characterises properties of that function.
- We leave it to the reader to find worlds and/or context associations for which ϕ does not hold.

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Models

- An interpretation $\mathcal{I}$ is a *model* for a formal system $\mathcal{L}$ if it evaluates all its axioms to true.
- An interpretation $\mathcal{I}$ is a *model* for a set of sentences Γ if it (the set) additionally evaluates all the sentences in Γ to true.
- The concept of model is a semantics notion.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Satisfiability, Entailment: $\models$ and Validity

- A set of sentences Γ is *satisfiable* if it (the set) has a model.
- A set of sentences Γ *entails* a sentence ψ

$$\Gamma \models \psi$$

- if every model of Γ is also a model of ψ,
- that is: ψ evaluates to true in every model of Γ.
- A sentence ψ is (universally) *valid*, and we write $\models \psi$, if it valuates to true in all models of its formal system.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Soundness and Completeness, $\vdash$ Versus $\models$

- A formal system is *sound* if $\Gamma \models \psi$ whenever $\gamma \vdash \psi$.
- Soundness helps ensure that every provable fact is true.
- A formal system is *complete* if $\Gamma \vdash \psi$ whenever $\gamma \models \psi$.
- Completeness helps ensure that every true fact is provable.
- Inconsistent systems cannot be sound.
- The formal systems used in the formal techniques for specification and verifying properties of specifications must be consistent, but are usually incomplete and not decidable.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- So the syntax (sentences, axioms and rules of inference) determines a proof theory.
- Issues like proofs, theorems, consistency and decidability are proof theoretic concepts.
- And an interpretation determines a model theory.
- Interpretations tie proof and model theories together.
- And so do issues like models, satisfiability, entailment, validity, soundness and completeness.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Topic 26
A Language of Boolean Ground Terms

- On one hand, we have the semantic notion of an algebra.
- And on the other hand, we have the syntactic notion of Boolean ground terms.
- The two together with appropriate syntactic and semantic extensions define a language of Boolean ground terms.
- In this lecture we will present these notions and extensions.

Syntax and Semantics

- The Boolean algebra to be put forward in these lectures can be presented as if it was an **RSL class**:

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

class Boolean
  type
    Bool
  value
    true, **false**, **chaos**
    $\sim$: **Bool** $\to$ **Bool**
    $\wedge$, $\vee$, $\Rightarrow$, $=$, $\neq$, $\equiv$: **Bool** $\times$ **Bool** $\to$ **Bool**
  axiom
    $\forall$ b,b′:**Bool** $\cdot$
      $\sim$b $\equiv$ **if** b **then false else true end**
      b $\wedge$ b′ $\equiv$ **if** b **then** b′ **else false end**
      b $\vee$ b′ $\equiv$ **if** b **then true else** b′ **end**
      b $\Rightarrow$ b′ $\equiv$ **if** b **then** b′ **else true end**
      b $=$ b′ $\equiv$ **if** (b$\wedge$b′)$\vee$($\sim$b$\wedge\sim$b′) **then true else false end**
      (b $\neq$ b′) $\equiv \sim$(b $=$ b′)
      (b $\equiv$ b′) $\equiv$ (b $=$ b′)
  end

.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- We emphasize that the above presents only an algebra:
  - its values
    - (by their designators **true, false, chaos**,
    - that is a semantic presentation)
  - and its operations
    - (by their signatures,
    - and by axioms defining the meaning of the operations).

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- And we emphasize that we have, in a sense, "misused" RSL. We can, of course, not use RSL to explain RSL. We are, above, informally using mathematics, but couch it in the style of some RSL-like text.
- In the next section we shall informally explain these operations.
- Later we shall introduce a language of Boolean ground terms by presenting the syntactic notions of grammar, axioms and rules of inference.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

The Connectives: $\sim$, $\wedge$, $\vee$, $\Rightarrow$, $=$, $\neq$, $\equiv$

- We explain the connectives, semantically, and as if we already allowed their operands to attain the undefined value **chaos**.
- For the algebra of Boolean ground terms we do not need the concept of 'undefined value'.
- Later we shall extend our logic to the language of predicate expressions, which have the same connectives as for Boolean ground terms.
- Below we therefore explain the connectives as if they occurred in propositional expressions, i.e., in truth-valued expressions whose variables were truth-valued.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Negation, ∼

- The logical connective ∼ is called 'negation'. We may read ∼P as 'not P'. The *law of the excluded middle* implies that we cannot have both 'not P' and 'P'; exactly one of the propositional expressions is true.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Conjunction, ∧

- The logical connective ∧ is called 'and' and 'conjunction'.
- The ∧ connective is applied not only to express the simultaneous truth of both operands,
- but also to express that if the left operand has truth value falsity, then one need not consider (evaluation of) the right operand!
- This non-commutativity of the ∧ connective cuts down on the size of expressions that one may need to write down:

  a ∧ b ≡ **if** a=**false then false else** b **end**

- The expression to the left of ≡ above is shorter than the expression to the right of ≡.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Disjunction, ∨

- The logical connective ∨ is called 'or', 'logical or', 'inclusive or' and 'disjunction'. Normally in the English language using 'or' means 'exclusive or' — for which latter exactly one of its two arguments are true, the other is false. But for P∨Q we accept if both are true. So beware!
- But if the left-hand operand is **true** then we may skip evaluating, i.e., even considering the right-hand operand.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Equality, =

- Equality, =, is to be seen in contrast to identity, ≡.
- In $E = E'$ the propositional expressions E and E' may contain arbitrary identifiers, i.e., variables, whose (in the present situation: truth) values may vary.
- Evaluation of $E = E'$ thus takes place in a context where these variables are bound to some values.
- And evaluation of $E = E'$ considers only the "current" context.
- That is, $E = E'$ may be evaluated several times, say because that expression occurs in a function definition body which is evaluated each time the function is invoked.
- The value of $E = E'$ is determined only by the context relevant for the specific invocation.
- For two different invocations the value of the same expression, $E = E'$, may thus differ!

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Implication, ⇒

- The logical connective ⇒ is called 'implication'. In P⇒Q the propositional expression P is called the *hypothesis*, the *antecedent* or the *premise*, while the propositional expression Q (of P⇒Q) is called the *consequence* or *conclusion*.
- The proposition P⇒Q is **false** only when P is **true** and (∧) Q is **false**.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- One can 'read' P⇒Q in a number of ways:
  - ⋆ *If P then Q*,
  - ⋆ *P only if Q*,
  - ⋆ *P is a sufficient condition for Q*,
  - ⋆ *Q is a necessary condition for P*,
  - ⋆ *Q if P*,
  - ⋆ *Q follows from P*,
  - ⋆ *Q provided P*,
  - ⋆ *Q is a logical consequence of P*, or
  - ⋆ *Q whenever P*.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Identity, ≡

- To explain the identity connective, ≡, is a bit more complicated than to explain the equality connective, =.
  - ⋆ As expressed above, when testing for equality of values one evaluates both operand expressions, once, in some current binding of their free identifiers to values,
  - ⋆ then tests them for equality.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- ⋆ For $\equiv(e',e'')$ (also written, more naturally, $e' \equiv e''$), one has to evaluate the two operand expressions in all possible bindings of their free identifiers to values, and for all bindings the same result must be yielded: Either always **true** or always **false** for the identity to hold, i.e., be **true**.
- ⋆ If some evaluate to **chaos**, then **chaos** is the value.
- ⋆ If none evaluate to **chaos** and not all to the same (**true** or **false**) truth value, then **false** is the value.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Three-Valued Logic

- The present material presents a proof-theoretic, i.e., a syntactic view of a three-valued logic of the emerging language of Boolean ground terms.
- Syntactically we should now present a set of axioms and, possibly, a set of rules of inference.
- We shall do so,
- but instead of presenting the rules of inference in the form of "something above a bar and something below that bar",
- we exemplify in a little while a *tabular representation* of these rules of inference.
- The **axiom**s are **true** and ∼**false**.
- But note that the above do not explain RSL in terms of RSL, but in terms of informal mathematics.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Syntactic Truth Tables

$\vee$	true	false	chaos
true	true	true	true
false	true	false	chaos
chaos	chaos	chaos	chaos

$\wedge$	true	false	chaos
true	true	false	chaos
false	false	false	false
chaos	chaos	chaos	chaos

$\Rightarrow$	true	false	chaos
true	true	false	chaos
false	true	true	true
chaos	chaos	chaos	chaos

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

≡ Versus =

- Assume e_1 and e_2 are defined expressions,
- both with deterministic (i.e., definite) values,
- without effects, that is, side effects (changes to assignable variables), and
- without communication, that is, as we shall first see in a much later lecture, CSP-like input/output communication.
- Assume further that e_1 and e_2 evaluates to v_1, and v_2, respectively.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Then the two three-valued logic truth tables are:

≡ and = Syntactic Truth Tables

≡	e1	e2	**chaos**
e1	**true**	**false**	**false**
e2	**false**	**true**	**chaos**
chaos	**false**	**false**	**true**

=	e1	e2	**chaos**
e1	**true**	**false**	**chaos**
e2	**false**	**true**	**chaos**
chaos	**chaos**	**chaos**	**chaos**

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Form of Inference Rule

- From the tabular form we arrive at the standard way of presenting a rule of inference

$$\frac{\text{antecedent(s)}}{\text{consequent}}$$

as follows:

  - ⋆ There is one rule of inference for each entry in each table.
  - ⋆ The antecedent of such a rule of inference is formed by composing three symbols:
    - ◇ the row index ground term,
    - ◇ the "upper left corner" operator, and
    - ◇ the column index ground term,

    and in that order.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

⋆ The consequent of the rule of inference is now the entry term:

$$\frac{\textbf{false} \Rightarrow \textbf{chaos}}{\textbf{true}}$$

- Above we have shown an example from the third table above, second row, third column!

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Truth and Falsity (Syntactic) Designators and Semantic Values

- As the truth tables are presented we may get the syntactic understanding that the truth designators are **true** and **false**.
- That is how we syntactically express them.
- Pragmatically we need a way to write down truth values — so we use the literals **true** and **false**.
- We distinguish between the syntactic literals — which are the ones we write down in our specifications — and the names of their meaning (i.e., semantics or interpretation).

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Some authors, when making this distinction, for example use the metalinguistic literals `tt`, `ff` and $\bot$.
- That is, the interpretation context (ρ) associates **true** with `tt`, etc.
- We could then use these latter as entries in three tables defining the interpretation context meaning of the connectives:

Interpretation Context: Semantic Truth Tables

$\vee$	`tt`	`ff`	$\bot$
`tt`	`tt`	`tt`	$\bot$
`ff`	`tt`	`ff`	$\bot$
$\bot$	$\bot$	$\bot$	$\bot$

$\wedge$	`tt`	`ff`	$\bot$
`tt`	`tt`	`ff`	$\bot$
`ff`	`ff`	`ff`	`ff`
$\bot$	$\bot$	$\bot$	$\bot$

$\Rightarrow$	`tt`	`ff`	$\bot$
`tt`	`tt`	`ff`	$\bot$
`ff`	`tt`	`tt`	`tt`
$\bot$	$\bot$	$\bot$	$\bot$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Non-commutativity of Boolean Connective

Let an expression be:

$$(\text{E1} \wedge \text{E2}) \vee \text{E3}$$

.

To express the above for commutative, two-valued logics of $\wedge$ and $\vee$, we need, for example, write:

if E1 **then** (**if** E2 **then true else** E3 **end**) **else** E3 **end**

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Ground Terms and Their Evaluation

Let us first give some examples:

Example 9.41 *Ground Terms:* Examples of ground terms are:

true, **false**, **~true**, **~false**,
true∧true, **true∨true**, **true⇒true**, **true=true**, **true≠true**, **true ≡ true**
true∧false, **true∨false**, **true⇒false**, **true=false**, **true≠false**, **true ≡ false**
...
(true∧((~true)∨false)⇒true)=false, ...

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Syntax of Boolean Ground Terms, BGT

- The *Boolean language of ground terms*, BGT, is now defined:
  - ⋆ The Basis Clause: **true, false** and **chaos** are Boolean ground terms.
  - ⋆ The Inductive Clause: If b and b′ are Boolean ground terms, then so are:
    - ◇ ∼b,
    - ◇ b∧b′, b∨b′, b⇒b′,
    - ◇ b=b′, b≠b′, b≡b′ and
    - ◇ (b).
  - ⋆ The Extremal Clause: Only those terms that can be formed from a finite number of uses of the above two clauses are Boolean ground terms.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- We can present the above inductive definition in the form of a BNF Grammar:

$$
\begin{aligned}
\langle \mathrm{BGT} \rangle \ ::= \ & \mathbf{true} \mid \mathbf{false} \mid \mathbf{chaos} \\
\mid \ & \sim \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle \wedge \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle \vee \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle \Rightarrow \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle = \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle \neq \langle \mathrm{BGT} \rangle \\
\mid \ & \langle \mathrm{BGT} \rangle \equiv \langle \mathrm{BGT} \rangle \\
\mid \ & (\ \langle \mathrm{BGT} \rangle \ )
\end{aligned}
$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- The trouble with the above grammar is that it is ambiguous.
- Is the term:

true $\wedge$ **false** $\vee$ **true**,

- the same as

true $\wedge$ (**false** $\vee$ **true**),

- or

(**true** $\wedge$ **false**) $\vee$ **true**?

- The inductive definition gave no hint as to the binding priority of the connectives.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- To do so, through a BNF grammar, we introduce an alternative grammar:

$$
\begin{array}{l}
\langle \mathrm{BGT} \rangle ::= \langle \mathrm{aBGT} \rangle \mid \langle \mathrm{pBGT} \rangle \\
\langle \mathrm{aBGT} \rangle ::= \mathbf{true} \mid \mathbf{false} \mid \mathbf{chaos} \\
\langle \mathrm{pBGT} \rangle ::= (\langle \mathrm{BGT} \rangle) \\
\quad \mid (\sim \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle \vee \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle \wedge \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle \Rightarrow \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle = \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle \neq \langle \mathrm{BGT} \rangle) \\
\quad \mid (\langle \mathrm{BGT} \rangle \equiv \langle \mathrm{BGT} \rangle)
\end{array}
$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Now it would not be possible to write:

  true $\wedge$ **false** $\vee$ **true**.

  $\star$ The above would have to be written either as

  true $\wedge$ (**false** $\vee$ **true**),

  $\star$ or as

  (**true** $\wedge$ **false**) $\vee$ **true**.

- By suitably designing a `BNF` grammar that directly "embodies" operator (binding) precedence rules, one can achieve an expression form that avoids excessive parenthesisation.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Boolean Ground Term Evaluation, Eval_BGT

- Given any Boolean ground term, we can provide an interpretation.
- That is, we can evaluate it.
- The evaluation rules are:
  - ⋆ If the ground term is **true**, its value is `tt`.
  - ⋆ If the ground term is **false**, its value is `ff`.
  - ⋆ If the ground term is ∼b and the value of b is `tt`, then the value of ∼b is `ff`.
  - ⋆ b value `ff` leads to ∼b result value `tt`.
  - ⋆ If the ground term is b∧b′ and the values of b and b′ are τ and τ' — where τ and τ', individually are one of `tt` or `ff` — then the value of b∧b′ is found by looking up under the corresponding entry in the ∧ table.
  - ⋆ The same holds for b⊙b′ where ⊙ is any of ∨, ⇒, =, ≠, or ≡, for which appropriate tables are selected.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- We "pseudo-formalise" this interpretation function.
- It is a pseudo-formalisation since it is not expressed in a proper formal notation.
- Why not, i.e., why not use RSL?
- The answer is: Because we have yet to introduce all the RSL machinery that is needed in a proper formalisation.
- The pseudo-formalisation shall serve to acquaint the reader with the form and possible content of formal function definitions.
- The tables are presented as maps (finite size, enumerable functions) from truth values to truth values.
- They are straightforward "mathematical" forms of the tables given above.
- One table was missing: that of negation. We leave it to the reader to provide that table.
- Thus the type of the Boolean ground term evaluation procedure, Eval_BGT, is:

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

```
value
  Eval_BGT: BGT → TBLS → Bool
type
  TBLS = uTBL×bTBL×bTBL×bTBL×bTBL×bTBL×bTBL
  uTBL = Bool -m→ Bool
  bTBL = Bool × Bool -m→ Bool
value
  Eval_BGT(bgt)(tbls) ≡
    let (n,a,o,i,eq,neq,id) = tbls in
    case bgt of
      true → tt,
      false → ff,
      chaos → ⊥,
      ∼t → let b = Eval_BGT(t)(tbls) in n(b) end,
      t′∧t″ →
        let b′=Eval_BGT(t′)(tbls), b″=Eval_BGT(t″)(tbls)
        in a(b′,b″) end,
      ... /∗ similar for p′∨p″, p′⇒p″, p′=p″, p′≠p″, and ∗/ p′is p″
    end end
```


Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

"Syntactic" Versus "Semantic Semantics"

- Thus there are two ways of looking at most of the languages that we will present in these lectures (for the various subsets of RSL, as well as for languages (or language fragments) separate from RSL).
  - ⋆ One way of looking at a language is *semantically* — as we have just done. Here we explained the meaning of (in this case Boolean ground) terms by exhibiting an evaluation procedure which "translated" the syntactic literals **true** and **false** into `tt`, respectively `ff`.
  - ⋆ Another way of looking at a language is *syntactically* — which we did earlier, for example on Foil 597. Then we basically "rewrote" an operand term in the Boolean literals **true** and **false** and connectives ($\sim$, $\wedge$, $\vee$, $\Rightarrow$, $=$, $\equiv$) into one of these literals.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- In the former semantics the meaning of a term was a mathematical value, one that "nobody has ever seen"!
- In the latter "semantics" the value of a term was a term, i.e., a syntactic "thing" that "everybody has seen"!
- The former style of semantics definition will be repeated, again and again in these lectures, and will be referred to, especially as we go on to the next examples, as the denotational style of semantics definitions.
- The latter style, the syntactic one, will be referred to as a 'rewrite rule' semantics. The λ-calculus, as given earlier was thus given a syntactical, that is, a rewrite rule semantics.
- "Syntactic semantics" is the basis
  - ⋆ for proofs of properties of formal specifications, and
  - ⋆ for proofs of certain relations (including correctness) between pairs of formal specifications.
- We shall return to this subject in due course.

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- We have introduced the "barest" of a language, the language of Boolean ground terms, BGT.
- We have separated our presentation into one of presenting the syntactics of BGT and one of presenting the semantics of BGT.
- And we have just, immediately above, briefly discussed a recurrent theme:
  - ⋆ a proper semantics view of syntax as well as
  - ⋆ a "syntactic semantics" as are most calculi.
- Finally, wrt. the pragmatics of BGT we said earlier:
  - ⋆ Using just the language of Boolean ground terms, there is not much of interest we can express.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- With the next logic language, that of propositions, there also is not much of interest we can express.
- We shall have to wait till we
  - ⋆ master the syntax (and semantics) of some language of predicates,
  - ⋆ then we can start expressing something.
- The reason for this seemingly slow, pedantic unfolding of two, we claim, not so "powerful" languages before we present the "real thing" is one of pedagogics and didactics:
  - ⋆ For some students the concepts of logics,
  - ⋆ and in particular its three "sublanguages", such as we have presented them,
  - ⋆ is not familiar.
  - ⋆ Additionally, the distinction between the syntactics of calculi (including proof systems) is so different from what they may be familiar with,
  - ⋆ that a direct, an immediate presentation of just a language of predicate calculus
  - ⋆ is an unnecessary intellectual challenge as compared to a stepwise unfolding such as we have attempted it.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Topic 27
Languages of Propositional Logic

- By *propositional logic* we syntactically understand
  - ⋆ (i) a set of truth values,
  - ⋆ (ii) an infinite set of *propositional expressions*, with connectives, and *truth-valued propositional variables*,
  - ⋆ (iii) a set of axioms and
  - ⋆ (iv) a set of rules of inference.
  - ⋆ The above determines a syntax, i.e., a proof theory of a propositional calculus.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Semantically we equip the (syntax of the language of) propositional logic with
  - ⋆ (v) a suitable context for determining the value of *propositional literals* and *symbols*, and
  - ⋆ (vi) an *interpretation function* that allows one to *calculate* the *truth value* of *propositional expressions*.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- By a propositional expression we thus mean an expression like a Boolean ground term, but where some Boolean literals (**true, false** or **chaos**) are replaced by *propositional variables.*
- A *propositional variable* is an identifier which, semantically, is intended to stand for a Boolean truth value (which could be **chaos**).
- We shall only cover propositional logic from the viewpoint of its practical use in formal specifications: (i–iv) Making precise the syntax of the expressions, and (v–vi) presenting an interpretation procedure for evaluating their values.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Propositional Expressions, PRO

Examples of Propositional Expressions

Let V be an alphabet of variable identifiers (i.e., variables), and let v, v′, ..., v″ be examples of such variables.

value v,v′,...,v″:**Bool**
... **true**, v, v∧**true**, ..., (∼(v∧v′)⇒(v′⇒v″)) = **false**, ...

The last line above exemplifies some propositional expressions.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Syntax of Propositional Expressions, PRO

- Basis Clause I: Any Boolean ground term is a *propositional expression.*
- Basis Clause II: There is given an *alphabet* V of (further un-analysed) *variable identifiers.* If v, v′, ..., v″ are in that alphabet, then v, v′, ..., v″ are propositional expressions.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- **Inductive Clause:** If p and p′ are propositional expressions, then so are
  - ⋆ ∼p,
  - ⋆ p∧p′,
  - ⋆ p∨p′,
  - ⋆ p⇒p′,
  - ⋆ p=p′,
  - ⋆ p≠p′,
  - ⋆ p≡p′ and
  - ⋆ (p).
- **The Extremal Clause:** Only such terms which can be formed from a finite number of uses of the above two clauses are propositional expressions.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- An example BNF grammar could be:

⟨PRO⟩ ::= **true** | **false** | **chaos**
| ∼ ⟨PRO⟩
| ⟨PRO⟩ ∧ ⟨PRO⟩
| ⟨PRO⟩ ∨ ⟨PRO⟩
| ⟨PRO⟩ ⇒ ⟨PRO⟩
| ⟨PRO⟩ = ⟨PRO⟩
| ⟨PRO⟩ ≠ ⟨PRO⟩
| ⟨PRO⟩ ≡ ⟨PRO⟩
| (⟨PRO⟩)
| ⟨ Identifier⟩
⟨ Identifier⟩ ::= ...

- We leave it to the student to complete the BNF definition of ⟨ Identifier⟩s, say as strings of alphanumeric characters commencing with lower case alphabetic characters, possibly having properly embedded, separating underscores (_).
- The above BNF grammar is ambiguous, as was the BNF grammar for Boolean ground terms, cf. Foil .

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Above we saw an example of an inductive definition.
- Next we shall see an example of a semantics which is presented in the style of a morphism, i.e., a homomorphism, such as earlier explained.
- The two concepts go hand-in-hand:
  - ⋆ The inductive definition describes composite structures in terms of postulated structures and operator symbols.
  - ⋆ A morphism is explained in terms of a function ϕ being applied to postulated (semantic) structures, i.e., values.
- The induction definition was here used to explain syntax.
- And homomorphisms will be used to explain the semantics of inductively, i.e., recursively, defined syntactic structures.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Examples

Example 9.42 ♣ *Propositions: 'Transportation Net':*
Let the following propositions be expressible:

- a: *Segment* 17 *of* *Broadway* *has connectors* *34th Street* *and* *35th Street*.
- b: *Segment* 18 *of* *Broadway* *has connectors* *35th Street* *and* *36th Street*.
- c: *Segment* 17 *of* *Broadway* *is connected to Segment* 18 *of* *Broadway*.

Given the above abbreviations we can express:

- $a \wedge b$, and $a \wedge b \Rightarrow c$,

If a and b holds then these propositions hold, i.e., c holds. ■

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Example 9.43 ♣ *Propositions: 'Container Terminal':*
Let the following propositions be expressible:

- a: *"Quay locations 7–12 are free at container terminal PTP."*
- b: *"The Harald Maersk ship is 6 terminal PTP quay locations long."*
- c: *"Harald Maersk can enter container terminal PTP."*

Given the above abbreviations we can express:

- $a \wedge b$, and $a \wedge b \Rightarrow c$,

If a and b holds then these propositions hold, i.e., c holds. ■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Example 9.44 ♣ *Propositions: 'Financial Service Industry':*
Let the following propositions be expressible:

- a: *Anderson has account α with a balance of US $ 1,000.*
- b: *Peterson has account π.*
- c: *Anderson can transfer US $ 200 from account α to Peterson account π.*

Given the above abbreviations we can express:

- $a \wedge b$, and $a \wedge b \Rightarrow c$,

If a and b holds then these propositions hold, i.e., c holds. ■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Proposition Evaluation, Eval_PRO

- To evaluate a propositional expression we must postulate a context function $\mathcal{C}$:

type
  $\mathcal{C} = \mathrm{V} \xrightarrow{m} \mathbf{Bool}$
value
  c:$\mathcal{C}$

- where $\mathcal{C}$ maps some, but not necessarily all, variables of any given propositional expression into a truth value.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The meaning of a propositional expression p, in the type of all propositional expressions PRO, is now a (function of type) partial function from contexts (i.e., $\mathcal{C}$) to Booleans!
- To see this, we show how to evaluate, how to find not the meaning, but the value of a propositional expression. And then we "lift" that value, that is, we abstract that propositional expression with respect to contexts, to obtain its meaning!

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- So, let some $c : \mathcal{C}$ be given, and postulate any propositional expression p.
  - ⋆ The value of any properly embedded Boolean ground term is found by the procedure outlined previously.
  - ⋆ If p is a variable v then the value of p is found by applying c to v, i.e., $c(\mathsf{v})$. If p, i.e., v, is not in the definition set of c, the result is the undefined value **chaos.**
  - ⋆ If p is a prefix expression $\sim\mathsf{p}'$, then first find the value, τ, of p', then negate it.
  - ⋆ If p is an infix expression $\mathsf{p}'\odot\mathsf{p}''$, then first find the values, τ', τ'' of p', respectively p''. Then proceed as for ground term evaluation.
  - ⋆ If p is a parenthesised expression (p'), then its value is that of the value of p'.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- This evaluation procedure will terminate since inductively (i.e., recursively) applied sub-evaluations apply to "smaller" and "smaller" subexpressions, and finally to ground terms and variables.
- The type of the propositional expression evaluation procedure is:

value

Eval_PRO: PRO → TBLS → $\mathcal{C} \xrightarrow{\sim}$ **Bool**

- The meaning of propositional expressions are therefore semantic functions $\mathcal{C}\xrightarrow{\sim}$**Bool**, while the value of a propositional expression is a **Bool**ean.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany


```
value
  Eval_PRO(pro)(tbls)(c) ≡
    case pro of
      true → tt,
      false → ff,
      chaos → ⊥,
      ∼p → let b = Eval_PRO(p)(tbl) in Eval_BGT(b)(tbls) end,
      p′ o p″ →
        let b′ = Eval_PRO(p′)(tbls)(c), b″ = Eval_PRO(p″)(tbls)(c)
        in Eval_BGT(b′ o b″)(tbls) end,
      (p) → Eval_PRO(p)(tbls)(c),
      v → c(v)
    end
```


Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Two-Valued Propositional Calculi

Preliminaries

- A propositional expression may evaluate to true for some (combinations of) values of its propositional variables, and to false for other (combinations of) values.
- A *tautology* is a propositional expression whose truth value is true for all possible values of its propositional variables.
- A *contradiction*, or *absurdity*, is a propositional expression which is always false.
- A propositional expression which is neither a tautology nor a contradiction is a *contingency*.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Some Proof Concepts

- An *assertion* is a statement. A *proposition* is an assertion which is claimed *true*.
- An *axiom* is a true assertion — typically about some mathematical structure. That is: axioms are *a priori* true; are not to be proven; cannot be proven; are not theorems.
- A *theorem* is a mathematical assertion which can be shown to be true.
- A *proof* is an argument which establishes the truth of the theorem.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- A proof of an assertion is a sequence of statements. The sequence of statements (re)presents an argument that the theorem is true.
  - ⋆ Some proof assertions may be *a priori* true:
    - ◇ Are either axioms or previously proven theorems.
  - ⋆ Other assertions may be *hypotheses* of the theorem — assumed to be true in the argument.
  - ⋆ Finally, some assertions may be *inferred* from other assertions which occurred earlier in the proof.
- Thus, to construct proofs, we need a means of drawing conclusions, or deriving new assertions from old ones.
- This is done by *rules of inference*.
- Rules of inference specify conclusions which can be drawn from assertions which are known, or can be assumed to be true.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Axioms and a Rule of Inference, I

- There are many ways of defining a propositional logic.
- First there is the issue of whether it is to be a two- or a three-valued logic, then there is the issue of which axioms and rules of inference to choose.
- Here we select a two-valued logic.
- Then we select a simple set of axioms and one rule of inference.
- Let ϕ, ψ and ρ designate metalinguistic variables.
- Any propositional expression may be put in their place.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- The following three axiom schemes are axioms of the chosen propositional calculus:

$$\phi \supset (\psi \supset \phi)$$
$$\phi \supset (\psi \supset \rho) \supset ((\phi \supset \psi) \supset (\phi \supset \rho))$$
$$(\sim (\sim (\phi))) \supset \phi$$

- There is a single rule of inference, *modus ponens:*

$$\frac{\phi, \phi \supset \psi}{\psi}$$

- Here we chose $\supset$ to designate implication.
- In the next example of a two-valued propositional logic we choose $\Rightarrow$ to designate implication.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- We can introduce additional connectives — other than $\neg$ (or $\sim$) and $\supset$ (or $\Rightarrow$) — through rules of inference.
- For example, disjunction ($\vee$): can be presented as:

$$\frac{\phi \vee \psi}{(\neg\phi) \supset \psi}, \quad \frac{(\neg\phi) \supset \psi}{\phi \vee \psi}$$

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Axioms and Inference Rules, II

- We shall now present another *formal proof system* allows proofs of propositional expressions to be fully done by machine. We can do this because there is only a finite number of propositional variables in any propositional expression, and each such variable's value ranges only over true or false, or is not defined at all, i.e., results in **chaos.**
- Here is a set of rules of inference for the propositional expressions of a twovalued logic. This set and those expressions, form a propositional calculus.
- Let ϕ, ψ, ρ, and ξ designate metalinguistic variables.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Substitution of equals for equals

- $\frac{\phi}{\phi \vee \psi}$ Addition

The form $\frac{\Phi}{\Psi}$ reads: From Φ conclude Ψ.

- $\frac{\phi \wedge \psi}{\phi}$ Simplification

- $\frac{\phi, \phi \Rightarrow \psi}{\psi}$,
  $\frac{\sim\psi, \phi \Rightarrow \psi}{\sim\phi}$ Modus Ponens versus Modus Tollens

The form $\frac{\Phi, \Psi}{\Omega}$ reads: From Φ and Ψ conclude Ω.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- $\frac{\sim\phi,\phi\vee\psi}{\psi}$,
  $\frac{\phi\Rightarrow\psi,\psi\Rightarrow\rho}{\phi\Rightarrow\rho}$ Disjunctive versus Hypothetical Syllogism

- $\frac{\phi,\psi}{\phi\wedge\psi}$ Conjunction

- $\frac{(\phi\Rightarrow\psi)\wedge(\rho\Rightarrow\xi),\phi\vee\rho}{\psi\vee\xi}$,
  $\frac{(\phi\Rightarrow\psi)\wedge(\rho\Rightarrow\xi),\sim\psi\vee\sim\xi}{\sim\phi\vee\sim\rho}$ Constructive vs. Destructive Dilemma

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- We have completed the second step of our unfolding of "the real thing": a language of predicates, calculus and interpretation.
- The structures of our presentation followed that of our previous presentation of the language of Boolean ground terms.
- The introduction of Boolean-valued identifiers, i.e., of propositional variables, is what distinguishes, syntactically, the language of Boolean ground terms from the language of propositions.
- Semantically these variables lead to a context which is expected to bind these variables to Booleans.
- But in order to make a logic language useful in dealing with actual world phenomena, there is also a need for allowing variables to designate other than Boolean values.
- To this we turn next.

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

Topic 28
Languages of Predicate Logic

- We now come to the "high point" of applied mathematical logic as far as this course is concerned.
- With predicate logic expressions of the kind that, for example, RSL allows us, we can express quite a lot.
- That is, predicate logic will be be a "work horse" for us.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Motivation

- In the propositional logics we cannot express the idea that *"if x is even then $x + 1$ is odd"*.
- To see this, let us carefully examine this statement.
- There are two independent propositions expressed here:
  - ⋆ $\mathsf{is\_even}(x)$ and
  - ⋆ $\mathsf{is\_odd}(\mathsf{succ}(x))$, where $\mathsf{succ}(x)$ yields the successor of x.
- The statement $\mathsf{is\_even}(x) \Rightarrow \mathsf{is\_odd}(\mathsf{succ}(x))$ is not a proposition.
  - ⋆ Its two terms are,
  - ⋆ but x is not a propositional variable, that is, one having a truth value.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- The *predicate calculus* extends propositional logic with *individual variables*, which model-theoretically may range over other than Boolean values,
- thus giving us the expressive power (in terms of quantifications)
- which allows us to express the above statement.
- For example:

$$\forall x : \mathbf{Int} \bullet \mathcal{O}(x) \Rightarrow \mathcal{E}(x+1)$$

where $\mathcal{O}$ and $\mathcal{E}$ designate the is_odd, respectively the is_even, predicates.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Informal Presentation

- By a *predicate logic* we syntactically, i.e., proof-theoretically, understand
  - ⋆ (i) a set of truth and other non-truth values;
  - ⋆ (ii) a usually infinite set of *predicate expressions* with
    - ◇ (ii.1) connectives,
    - ◇ (ii.2) *truth-valued propositional variables*,
    - ◇ (ii.3) *usually other non-truth-valued quantified* or *free variables*,
    - ◇ (ii.4) *quantified expressions;*
  - ⋆ (iii) a set of axiom schemes; and
  - ⋆ (iv) a set of rules of inference.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Semantically, i.e., model-theoretically, we understand a predicate calculus to extend the above with:
  - ⋆ (v) for every predicate expression, a context, $c : \mathcal{C}$, which maps individual variables to values, and
  - ⋆ (vi) an interpretation procedure for determining, given any context and any predicate expression, the value of that expression.
- Predicate expressions are thus extensions of propositional expressions: Where a propositional expression may occur, it now becomes possible to express a property by expressing some truth-valued relations between other than truth values.

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.45 *Predicate Expressions:*

- Informally, an example is:

$$((\mathsf{e}-1 \leq 3) \Rightarrow \mathsf{e}') \Rightarrow (\exists\ \mathrm{i}{:}\mathbf{Int} \cdot \mathrm{i} > \mathsf{e} * (\mathsf{e}'' + 3))$$

- which we can read: if $\mathsf{e}-1$ less than or equal to 2 *implies* e' then that *implies* that there *exists* an *integer* which is *larger* than the *value* of the non-truth valued expression $\mathsf{e} * (\mathsf{e}'' + 3)$.
- The example illustrates a number of new constructs that — from now on — may occur in logical, i.e., *predicate* expressions.

$$\leq,\ >,\ \exists,\ -,\ *,\ +$$

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- More generally, and in this case schematically.
- we can list the constructs of a predicate calculus:

[1] p(e,e′,...,e″)
[2] f(t,t′,...,t″)
[3] ∀ x:X•E(x)
[4] ∃ x:X•E(x)
[5] ∃ ! x:X•E(x)

- which we can read semantically:

1 The formula $\mathsf{p(e,e',\ldots,e'')}$ expresses the holding, or non-holding of some relation, p, between the values of subexpressions e, $\mathsf{e'}$, ..., $\mathsf{e''}$.

2 The value of expression $\mathsf{f(t,t',\ldots,t'')}$ is the result of applying the non-truth result valued function, f, to the values of subexpressions t, $\mathsf{t'}$, ..., $\mathsf{t''}$.

3 For all values x of type X it is the case that $\mathsf{E(x)}$ holds.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

4 There exists at least one value x of type X for which it is the case that E(x) holds.

5 There exists a single, unique value x of type X such that E(x) holds.

- Whether these predicate expressions ([1-5]) hold, i.e., are **true** or not (**false** or **chaos**) is not guaranteed just by writing them!
- Forms [3-4-5] illustrated the concepts of binding and typing, $x : X$:
  - ⋆ A typing is, generally, a clause of either of the forms:

    identifier : type_expression
    identifier_1,identifier_2,...,identifier_n : type_expression

  - ⋆ Typings bind their identifier[_i]s to (arbitrary) values of the type designated by the type_expression.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Examples

Example 9.46 ♣ *Predicates: 'Transportation Net':*

- Assume that from nets, $n : N$,
  - ⋆ we can observe segments, $s : S$, and connections, $c : C$,
  - ⋆ and that from segments [respectively connections] we can observe connection identifiers [respectively segment identifiers],
  - ⋆ then we must assume that the latter observations fit with the former:
    - ◇ That all segments [respectively connections] of the net have unique identifiers, and
    - ◇ that any segment [respectively connection] identifier observed from a connection [respectively segment]
    - ◇ is the identifier of a segment [respectively connection] observed in the net.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

type
  N, S, C, Si, Ci
value
  obs_Ss: N → S-**set**
  obs_Cs: N → C-**set**
  obs_Sis: (N|C) → Si-**set**
  obs_Cis: (N|S) → Ci-**set**
axiom
  ∀ n:N •
    card obs_Ss(n) = **card** obs_Sis(n) ∧
    card obs_Cs(n) = **card** obs_Cis(n) ∧
    ∀ s:S • s ∈ obs_Ss(n) ⇒ obs_Cis(s) ⊆ obs_Cis(n) ∧
    ∀ c:C • c ∈ obs_Cs(n) ⇒ obs_Sis(c) ⊆ obs_Sis(n)

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- The first axiom clause expresses uniqueness of identifiers: the cardinality of segments [respectively connections] and segment [respectively connection] identifiers are the same.
- If you do not like that form, then try this instead:

type
  N, S, C, Si, Ci
value
  obs_Ss: N → S**-set**
  obs_Cs: N → C**-set**
  obs_Si: S → Si
  obs_Ci: C → Ci
axiom
  ∀ n:N •
    ∀ s,s′:S • {s,s′} ⊆ ∈ obs_Ss(n) ∧ s≠s′ ⇒ obs_Si(s) ≠ obs_Si(s′) ∧
    ∀ c,c′:C • {c,c′} ⊆ ∈ obs_Cs(n) ∧ c≠c′ ⇒ obs_Ci(c) ≠ obs_Ci(c′)

■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.47 ♣ *Predicates: 'Container Terminal':*

- Assume that from container terminals, ct:CT, we can observe
  - ⋆ (i) the container storage area, csa:CSA and
  - ⋆ (ii) containers, c:C (in the container storage area).
- That from the former we can observe
  - ⋆ (iii) bays, bay:Bay,
  - ⋆ (iv) rows, row:Row,
  - ⋆ and (v) stacks, stk:Stk,
  - ⋆ and that from any of these (bays, rows and stacks) one can observe containers.
  - ⋆ Finally assume that from the latter we can observe (vi) containers, c:C:

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

type
CT, C, CSA, BAY, ROW, STK
value
obs_Cs: (CT|CSA|BAY|ROW|STK) → C**-set**
obs_CSA: CT → CSA
obs_BAYs: (CT|CSA) → BAY**-set**
obs_ROWs: (CT|CSA|BAY) → ROW**-set**
obs_STKs: (CT|CSA|BAY|ROW) → STK**-set**

- Now containers observed in the container terminal
- must be containers
  - ⋆ of some unique stack,
  - ⋆ of some unique row and
  - ⋆ of some unique bay
- of the container storage area:

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

axiom

$$
\begin{array}{l}
\forall\ \mathrm{ct{:}CT} \cdot \\
\quad \forall\ \mathrm{c{:}C} \cdot \mathrm{c} \in \mathrm{obs\_Cs(ct)} \Rightarrow \\
\quad\quad \textbf{let}\ \mathrm{csa} = \mathrm{obs\_CSA(ct)}\ \textbf{in} \\
\quad\quad \exists!\mathrm{bay{:}BAY} \cdot \\
\quad\quad\quad \mathrm{bay} \in \mathrm{obs\_BAYs(csa)} \wedge \mathrm{c} \in \mathrm{obs\_Cs(bay)} \\
\quad\quad\quad \Rightarrow \exists!\mathrm{row{:}ROW} \cdot \\
\quad\quad\quad\quad \mathrm{row} \in \mathrm{obs\_ROWs(bay)} \wedge \mathrm{c} \in \mathrm{obs\_Cs(row)} \\
\quad\quad\quad\quad \Rightarrow \exists!\mathrm{stk{:}STK} \cdot \\
\quad\quad\quad\quad\quad \mathrm{stk} \in \mathrm{obs\_STKs(row)} \wedge \mathrm{c} \in \mathrm{obs\_Cs(stk)} \\
\quad\quad \textbf{end}
\end{array}
$$

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Example 9.48 ♣ *Predicates: 'Financial Service Industry':*

- Assume that from a bank, bank:Bank, one can observe
  - ⋆ (i) the unique identities, cid:Cid, of all its customers,
  - ⋆ (ii) the unique identities, aid:Aid, of all their accounts,
  - ⋆ (iii) the collection, accs:Accs, of all these accounts,
  - ⋆ (iv) the identities of all the accounts, acc:Acc, in the collection, accs:Accs, of all accounts,
  - ⋆ (v) the account numbers owned by any one identified customer
  - ⋆ and (vi) the identities of customers possibly sharing any one (identified) account.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

type
Bank, Cid, Aid, Accs, Acc
value
obs_Cids: Bank → Cid**-set**
obs_Aids: (Bank|Accs|(Bank×Cid)) → Aid**-set**
obs_Accs: Bank → Accs
obs_Cids: Bank × Aid → Cid**-set**

- (vii) If a customer is registered in a bank then we assume that customer to have one or more accounts.
- (viii) If an account is known by the bank then it is an account in the collection of accounts.
- (ix) And if that account is shared by one (!) or more customers then they are all known to the bank and as having that account.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

axiom

$$
\begin{array}{l}
\forall\ \text{bank:Bank} \cdot \\
\quad \forall\ \text{cid:Cid} \cdot \text{cid} \in \text{obs\_Cids}(\text{bank}) \Rightarrow \\
\qquad \text{obs\_Aids}(\text{bank},\text{cid}) \neq \{\} \wedge \\
\quad \forall\ \text{aid:Aid} \cdot \text{aid} \in \text{obs\_Aids}(\text{bank}) \Rightarrow \\
\qquad \text{aid} \in \text{obs\_Aids}(\text{obs\_Accs}(\text{bank})) \wedge \\
\qquad \forall\ \text{cid}',\text{cid}'':\text{Cid} \cdot \\
\qquad\quad \text{cid}' \in \text{obs\_Cids}(\text{bank},\text{aid}) \Rightarrow \\
\qquad\qquad \text{cid}' \in \text{obs\_Cids}(\text{bank}) \wedge \text{aid} \in \text{obs\_Aids}(\text{bank},\text{cid}')
\end{array}
$$

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Quantifiers and Quantified Expressions

Syntax

- Quantified expressions, like ∀x:X·E(x), ∃x:X·E(x) and ∃!x:X·E(x), are *predicate expressions.*
- In general, *quantified expressions* are of the *inductive* form:
  - ⋆ Let x be any identifier, let X be any type expression,
  - ⋆ and let E(x) be
    - ◇ any *propositional* or *predicate expression*
    - ◇ in which x may (or may not) *occur*,
    - ◇ and if it *occurs*, may *occur free* or *bound.*
  - ⋆ Now ∀x:X·E(x), ∃x:X·E(x) and ∃!x:X·E(x), are *quantified predicate expressions.*
- The *extremal clause* follows.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- We refer to the above ∀, ∃ and ∃! as *quantifiers*, to x's as *binding variables*, E(x) as the *body* of the *quantified expression*, and to X as the *range set* (designated by a type expression) of the *quantification*.

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

Free and Bound Variables

- In the λ-calculus we define a concept of *free* and *bound* variables.
  - ⋆ Let E(x) be an expression
    - ◇ which is not of the form $\mathcal{Q}$x:X·E(x),
    - ◇ where $\mathcal{Q}$ is either of $\forall$, $\exists$ or $\exists!$,
    - ◇ and in which there are no further embedded, i.e., proper subexpressions of those forms,
    - ◇ then any occurrence of x in E(x) is *free*.
  - ⋆ Let E(x) be an expression
    - ◇ which is of the form $\mathcal{Q}$x:X·E(x),
    - ◇ where $\mathcal{Q}$ is either of $\forall$, $\exists$ or $\exists!$,
    - ◇ then any occurrence of x in E(x) is *bound*.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- ⋆ Let E(x) be an expression
  - ◇ which is not of the form $\mathcal{Q}$x:X·E(x),
  - ◇ where $\mathcal{Q}$ is either of ∀, ∃, or ∃!,
  - ◇ but in which there are some further embedded, i.e., proper sub-expressions of those (x binding) forms,
  - ◇ then any occurrence of x in E(x), which is not within those latter forms, is *free*, whereas, of course,
  - ◇ the others are *bound*.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Compound Quantified Expressions

- Since in $\mathcal{Q}$x:X·E(x) the expression *body* may itself be of the form $\mathcal{Q}$y:Y·E′(y), we may get multiple *bindings:*

  ... ∀ x:X · ∀ x′:X · ∃ y:Y · ∀ z:Z · E(x,x′,y,z)

- for which we provide a shorthand:

  ... ∀ x,x′:X, z:Z, ∃ y:Y · E(x,x′,y,z)

Example 9.49 *Compound Predicate Expression:* For all natural numbers i larger than 2 there exist two distinct natural numbers j, k larger than 0 (but not necessarily distinct from i) such that i is the product of j and k:

∀ i:**Nat** · i>2 ⇒ ∃ j,k:**Nat** · j≠k ∧ i = j∗k

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

Example 9.50 *Compound Predicate Expression:* For all sets s of integers such that if i is in the set then also $-i$ is in the set; it is the case that the sum of all integers equals 0.

type
  S = **Int-set**
value
  sum: S → **Int**
  sum(s) ≡
    if s={} **then** 0 **else let** i:**Int** · i ∈ s **in** i + sum(s\{i}) **end end**
axiom
  ∀ s:S · ∀ i:**Int** · i ∈ s ⇒ −i ∈ s ⇒ sum(s) = 0

Here **Int-set** designate the type all of whose values are sets of integers. ■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Syntax of Predicate Expressions, PRE

The Symbols of a Predicate Calculus

- The symbols of a predicate calculus include a number of elements.
- There are the variables b, b', ..., b'' and x, x', ..., x'', where we think of the b's being truth valued propositional variables, and the x's being otherwise typed variables (integers, etc.).
- There are the Boolean connectives $\sim$, $\vee$, $\wedge$, etc.
- There are the existential quantifiers $\exists$, $\exists!$ and $\forall$.
- For every suitable arity n there are sets of predicate function symbols $\{p_{n_1}, p_{n_2}, \ldots, p_{n_{p_n}}\}$.
- For every suitable arity m there are sets of otherwise typed function symbols $\{f_{m_1}, f_{m_2}, \ldots, f_{m_{f_m}}\}$.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

The idea is that:

$$p_{i_j}(t_1, t_2, \ldots, t_i), j : 1, 2, \ldots, i_p;$$

and

$$f_{k_\ell}(t'_1, t'_2, \ldots, t'_k), \ell : 1, 2, \ldots, i_f;$$

are two expression forms.

- The first is a *formula* and ostensibly has a truth value; that
- the second is a *term* and ostensibly has a value of any kind (i.e., of any type).
- Finally the arguments $t_j, t'_{j'}$ are also *terms* of any kind (i.e., of any type) of value.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Note

- that we now distinguish between *terms* as the basic building blocks of expressions,
- and *formulas* as the expressions that have truth values.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

The Term Language of a Predicate Calculus

- The term language is defined inductively:
  - ⋆ **Basis Clause:** A variable, b, etc., or x, etc., that is, whether truth valued or not, is a *term.*
  - ⋆ **Inductive Clause:** If $t_1, t_2, \ldots, t_n$ are terms and f_n is an n-ary function symbol, and if p_n is an n-ary predicate symbol then, $f_n(t_1, t_2, \ldots, t_n)$ and $p_n(t_1, t_2, \ldots, t_n)$ are terms.
  - ⋆ **Extremal Clause:** Only those expressions that can be formed from a finite number of applications of the above clauses are terms.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

- The idea is that
  - ⋆ Boolean literals are nullary predicate function symbols: true() ≡ **true**, false() ≡ **false** and chaos() ≡ **chaos**; and that,
  - ⋆ for example, numerals are nullary function symbols: one() ≡ 1, etc.
- More complex examples are:
  - ⋆ and(b, b') ($\equiv b \wedge b'$), etc.; and
  - ⋆ ift(equalzero(i),one(),mult(i,fact(sub(i,1)))) (≡ **if** i=0 **then** 1 **else** i×fact(i-1) **end**), etc.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

The Atomic Formula Language of a Predicate Calculus

- The atomic formula language is defined inductively:
  - ⋆ **Basis Clause:** Any propositional expression is an atomic formula (and is a term).
  - ⋆ **Inductive Clause:** If $t_1, t_2, \ldots, t_n$ are terms and p_n is an n-ary predicate function symbol, then $p_n(t_1, t_2, \ldots, t_n)$ is an *atomic formula.* [15]
  - ⋆ **Extremal Clause:** Only such terms which can be formed from a finite number of uses of the above two clauses are atomic formulas.

[15] The truth and the non-truth-value relational operators (to wit: $=, \neq, \equiv, \not\equiv, <, \leq, >, \geq$, etc.) are examples of p_2's, and hence of atomic formulas, as would be any user-defined predicate applied to terms.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

The Well-formed Formulas of a Predicate Calculus

- The wff language is defined inductively:
  - ⋆ Basis Clause: *Atomic formulas* are *formulas*, i.e., *predicate expressions*.
  - ⋆ Inductive Clause: If x is a variable ranging over type X, and u, v and $\mathcal{E}(x)$ are formulas (i.e., predicate expressions), then:
    - ◇ $\sim u$ is a formula;
    - ◇ $u \wedge v$, $u \vee v$, $u \Rightarrow v$, $u = v$, $u \neq v$, and $u \equiv v$, are formulas;
    - ◇ $\forall x{:}X \cdot \mathcal{E}(x)$, $\exists x{:}X \cdot \mathcal{E}(x)$, and $\exists! x{:}X \cdot \mathcal{E}(x)$ are formulas.
  - ⋆ Extremal Clause: Only those terms that can be formed from a finite number of uses of the above two clauses are formulas, i.e., predicate expressions.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

An Informal BNF Grammar for Predicate Expressions

⟨Fn⟩ ::= ⟨Identifier⟩ /∗ Fn: non−truth valued functions ∗/
⟨Pn⟩ ::= ⟨Identifier⟩ /∗ Pn: truth valued predicates ∗/
⟨Term⟩ ::= ⟨Identifier⟩
| ⟨Fn⟩ (⟨Term−seq⟩)
| ⟨Pn⟩ (⟨Term−seq⟩) /∗ **true, false, chaos**: nullary terms ∗/
⟨Term−seq⟩ ::= /∗ empty sequence ∗/
| ⟨Term⟩
| ⟨Term⟩ ⟨Comma−Term−seq⟩
⟨Comma−Term−seq⟩ ::= ⟨Comma−Term⟩ ⟨Term−seq⟩
⟨Comma−Term⟩ ::= , ⟨Term⟩
⟨Atom⟩ ::= ⟨Identifier⟩ /∗ Boolean valued ∗/
| ⟨Pn⟩ (⟨Term−seq⟩)
⟨Wff⟩ ::= ⟨Atom⟩
| ∼ ⟨Wff⟩
| ⟨Wff⟩ ∧ ⟨Wff⟩ | ⟨Wff⟩ ∨ ⟨Wff⟩ | ⟨Wff⟩ ⇒ ⟨Wff⟩
| ⟨Wff⟩ = ⟨Wff⟩ | ⟨Wff⟩ ≠ ⟨Wff⟩ | ⟨Wff⟩ ≡ ⟨Wff⟩
| ⟨Quant⟩ ⟨Identifier⟩ : ⟨Tn⟩ • ⟨Wff⟩
⟨Quant⟩ ::= ∃ | ∃ ! | ∀

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

A Predicate Calculus

Axiom Schemes

- Let $\phi[x \mapsto t]$ designate the expression ϕ' which is like ϕ except that some or all of the free x in ϕ have been replaced by the term t — where x does not occur free in t.
- One such system for the predicate calculus extends one, or the other of the sets of axiom schemes given (earlier) for a propositional calculus with the following:

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- *Provided that no free occurrence of x in ϕ lie in the scope of any quantifier for a free variable appearing in the term t,* we have:

$$\forall x : X \bullet \phi(x) \Rightarrow \phi[x \mapsto t]$$

Expressed semantically: *If some formula ϕ is true for all x, then it is certain true when some particular term t is substituted for x in ϕ.*

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- And, *provided that* t *is free for* x *in* ϕ, we have:

$$\phi[x \mapsto t] \Rightarrow (\exists x : X \bullet \phi(x))$$

Expressed semantically: *We can conclude that there exists some* x *satisfying the formula* ϕ *if some substitution instance of* ϕ *is true.*

Rules of Inference

The above leads to the following rules of inferences:

- First:

$$\frac{\psi \supset \phi(v)}{\psi \supset (\forall x : X \bullet \phi(x))},$$

- *and:*

$$\frac{\phi(v) \supset \psi}{(\exists x : X \bullet \phi(x)) \supset \psi},$$

where the variable v is not free in ψ.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- *The rule of universal quantification can best be understood* semantically *by considering the simpler case when* ψ *is true. Then the rule becomes:*

$$\frac{\phi(v)}{\forall x : X \bullet \phi(x))}$$

which, semantically *says, that if* ϕ *is true for some arbitrary* v*, then it must be true for all* x*.*

- *Universal and existential quantification are related:*

$$\exists x : X \bullet \phi(x) \equiv \sim (\forall x : X \bullet \sim \phi(x)))$$

This definition, as an axiom, can be done if we have already defined equivalence.

Predicate Expression Evaluation

Evaluation Contexts

- Semantically we may understand the predicate calculus by constructing models.
- There are two parts to any such model:
  - ⋆ a context, $\rho : \mathcal{R}$, which maps all user-defined symbols in the language of predicate expressions to their meaning in some world Ω,
  - ⋆ and an interpretation function.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Thus, in order to find the value of a given predicate expression, one must provide a context which maps
  - ⋆ some, all or more of the *free variables*, v:V (of that predicate expression), into values, VAL, of appropriate types;
  - ⋆ some, all or more of the *type names*, Tn (of the range type [name] expressions of that predicate expression), into their respective — finite or even infinite value spaces;
  - ⋆ some, all or more of the predicate function symbols, p (of that predicate expression), into appropriate arity predicate functions; and
  - ⋆ some, all or more of the non-truth result value function symbols, f (of that predicate expression), into appropriate arity non-truth result value functions:

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

type

Vn, Tn, Pn, Fn, VAL

$\mathcal{R}$ = (Vn→VAL)
  ∪ (Tn→VAL-**set**)
  ∪ (Pn→(VAL* → **Bool**))
  ∪ (Fn→(VAL* → VAL))

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Recall that $A \to B$ stands for the type whose values are functions from A into B,
- that A**-set** stands for the type whose values are sets of element values of type A and
- that A^* stands for the type whose values are lists of element values of type A.
- The unusual, non-`RSL` construct $(A \to B) \cup (C \to D)$ stands for the type whose values are functions from A into B and functions from C into D.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.51 *Predicate Expression Evaluation Context:*

- Let us review an example. See the first formula line below.
- To evaluate the next expression we seem to need a context, $c : \mathcal{C}$, like the one shown further below:

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany


```
value
  (a ∧ (v ≥ 7)) ⇒ ∀ k:K • fact(j) ≤ k

  ρ: λx:(Vn|Tn|Pn|Fn) •
       if x ∈ Vn then
         case x of
           a→t,v→i,j→m, ...
         end
       else if x ∈ Tn then
         case x of K→{−2,−1,0,1,2}, ... end
       else if x ∈ Pn then
         case x of
           "larger-than-or-equal" → λ(x,y):(Int×Int)•x≥y,
           "smaller-than-or-equal" → λ(x,y):(Int×Int)•x≤y,
           ...
         end
       else /∗ assert: ∗/ x ∈ Fn:
         case x of
           "factorial" → λn:Int•if n=0 then 1 else n∗fact(n−1) end,
           ...
         end
       end end end
```


Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- As an example, let
- $(\mathsf{a} \wedge (\mathsf{v} \geq 7)) \Rightarrow \forall \mathsf{k}{:}\mathsf{K} \cdot \mathsf{fact}(\mathsf{j}) \leq \mathsf{k}$
- be the predicate expression to be evaluated.
- Variables a, v and j are free and so is type name K — the latter is assumed to be some (finite or infinite) set of integers.
- For that expression we need a context preferably like $\rho : \mathcal{R}$ above — where t is some Boolean truth value, and i and m are some integers.
- If the values of t, i, m are **true,** $9, -2$ then we see that the predicate evaluates to **true**.

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Meaning Versus Values of Predicate Expressions

- The meaning of a predicate expression p, in the type of all predicate expressions PRE, is now a function from context, that is, $\rho : \mathcal{R}$ to Booleans!

  value

  Eval_PRE: PRE $\rightarrow \mathcal{R} \xrightarrow{\sim}$ **Bool**

- To see this, we show how to evaluate — how to find — not the meaning, but the value of a predicate expression. And then we "lift" that value, we abstract that predicate expression with respect to contexts!

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Evaluation Procedure, Eval_PRE

• Term Evaluation

- Let $\rho : \mathcal{R}$ be some context, and let t be the term subject to evaluation in context ρ.
- If t is variable v then
  - ⋆ c is applied to v to find its value.
  - ⋆ If v is not in the definition set of ρ then the undefined value **chaos** is yielded.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If, instead, t is of the form f(t,t′,...,t″), then
  - ⋆ the values v, v′, ..., v″ of the terms t, t′, ..., t″, respectively, are evaluated;
  - ⋆ the function f is "looked up" in ρ (i.e., c(f)),
  - ⋆ and the resulting function ψ applied to v, v′, ..., v″: ψ(v,v′...,v″).
  - ⋆ If f is not in the definition set of ρ, then the undefined value **chaos** is yielded.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

• Formula Evaluation

- Let e be a formula.
- If e is a propositional expression, that is,
  - ⋆ if e is of any of the forms: ∼e, e∧e′, e∨e′, e=e′, e≠e′, or e≡e′,

then evaluate as prescribed earlier (Eval_pro).

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- If e is of the form p(t,t′,...,t″) then
  - ⋆ the values v, v′, ..., v″ of the terms t, t′, ..., t″, respectively, are evaluated,
  - ⋆ the predicate function p is "looked up" in c (i.e., ρ(p)),
  - ⋆ and the resulting function ϕ applied to v, v′, ..., v″: ϕ(v,v′...,v″).
  - ⋆ If p is not in the definition set of c, then the undefined value **chaos** is yielded.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If, instead, e is of either of the forms ∀x:X·E(x), ∃x:X·E(x) or ∃!x:X·E(x), i.e., if it is of the general form $\mathcal{Q}$ x:X·E(x), then
  - ⋆ the value, Ξ, of the range set X is found from ρ.
  - ⋆ If X is not in the definition set of ρ, then the undefined value **chaos** is yielded, and becomes the value of $\mathcal{Q}$ x:X·E(x).
- Otherwise three case distinctions must be made:

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If $\mathcal{Q}$ is $\forall$ then the possibly infinite conjunction: $\mathsf{E}(\xi_1)\wedge\mathsf{E}(\xi_2)\wedge...\wedge\mathsf{E}(\xi_i)\wedge...$ is evaluated.
  - ⋆ Here the ξ's range over all, possibly infinite values of Ξ.
  - ⋆ **Note:** *The $\wedge$ is here constrained to be commutative.*
  - ⋆ All $\mathsf{E}(\xi_i)$ must yield **true** for $\forall$ x:X·E(x) to yield **true.**
  - ⋆ Any **chaos** results in **chaos.**
  - ⋆ Any **false** with no **chaos** yields **false** for $\forall$ x:X·E(x).
  - ⋆ We can rephrase the above:
    - ◇ The value of $\forall$x:X·E(x) is **true** if E(x) holds for all models as implied by x:X.
    - ◇ That is, x:X defines a set of models, that is, a set of contexts, at least one for each element x in X.
    - ◇ Each of these models further defines bindings of all other free identifiers in E(x).

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If $\mathcal{Q}$ is $\exists$ then there must exist a disjunction:

$$\mathrm{E}(\xi_1)\vee\mathrm{E}(\xi_2)\vee...\vee\mathrm{E}(\xi_i)\vee...$$

This disjunction is evaluated.

  - ⋆ For it to yield **true** $\mathsf{E}(\xi_1)$ must yield **true**
  - ⋆ with all other $\mathsf{E}(\xi_j)$ for all $\mathsf{j}>1$ yielding **true, false** or **chaos.**
  - ⋆ Or rephrased: $\exists \mathsf{x}:\mathsf{X}\cdot\mathsf{E}(\mathsf{x})$ is true if $\mathsf{E}(\mathsf{x})$ holds for at least one model in the set of models induced by X.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- If $\mathcal{Q}$ is $\exists!$
  - ⋆ then there must exist exactly one i in some arbitrary disjunction:

    $$\mathrm{E}(\xi_1)\vee\mathrm{E}(\xi_2)\vee...\vee\mathrm{E}(\xi_i)\vee...$$

  - ⋆ such that $\mathsf{E}(\xi_1)$ yields **true** and all other $\mathsf{E}(\xi_i)$, for all $\mathsf{i}>1$, yield **false** or **chaos**!
  - ⋆ Rephrased: $\exists!\mathsf{x}{:}\mathsf{X}{\cdot}\mathsf{E}(\mathsf{x})$ holds if and only if $\mathsf{E}(\mathsf{x})$ holds for exactly one of the induced models.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

First-Order and Higher-Order Logics

- If the *range set* of quantifications permit values that are, or contain, functions, then we say that the *predicate logic* is a *higher-order logic*.
- Otherwise it is a *first-order logic*.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- An example may be in order to illustrate the need for higher-order logics:

type
  P = A → **Bool**
value

axiom
  ∀ p:P · ...

- RSL's logic is higher-order.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Validity, Satisfiability and Models

Contexts and Interpretations

- We have seen that predicate expressions only have values if a suitable context is given.
- In mathematical logic such a context is called an interpretation.
- Generally a context, that is, an interpretation, is a mapping of identifiers to mathematical values.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- ⋆ Predicate symbols p_n of arity n can be thought of as being mapped $(p_n \mapsto \pi)$ into possibly infinite sets π of n groupings: $(v_1, v_2, \ldots, v_n)$, $p_n(v_1, v_2, \ldots, v_n)$ represents truth for all $(v_1, v_2, \ldots, v_n)$ in π, and falsity otherwise.
- ⋆ Function symbols f_n of arity n can likewise be thought of as being mapped $(f_n \mapsto \phi)$ into possibly infinite sets, ϕ of $n+1$ groupings: $(v_1, v_2, \ldots, v_n, v)$ — $f_n(v_1, v_2, \ldots, v_n)$ has value v for respective $(v_1, v_2, \ldots, v_n, v)$ in ϕ, and is otherwise undefined.
- ⋆ Nonfunction symbols, i.e., variable identifiers, i are mapped $(i \mapsto v)$ into values v.

Example 9.52 *Predicate Expression Interpretation:* An example may be in order. We interpret the predicate ... ∀ i:Integer, ∃ n:Natural · square(i) = n ... in two models:

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

type
Integer, Natural
value
square: Integer → Natural
... ∀ i:Integer, ∃ n:Natural · square(i) = n ...

/* interpretation_1: */
[Integer↦{ ...,−2,−1,0,1,2,... },
Natural↦{ 0,1,2,... },
square↦{ ...,(−2,4),(−1,1),(0,0),(1,1),(2,4),... }]

/* interpretation_2: */
[Integer↦{ ...,−2,−1,0,1,2,... },
Natural↦{ 0,3,5,7,9,... },
square↦{ ...,(−2,4),(−1,1),(0,0),(1,1),(2,4),... }]

The above predicate is true in interpretation_1 and false in interpretation_2. ■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Validity and Satisfiability

- Let there be given a possibly infinite set of interpretations.
- A predicate expression is said to be *valid* if it is true for all interpretations.
- A predicate expression is said to be *satisfied* if it is true for at least one interpretation.
- There is no mechanical procedure by which one can determine the validity or satisfiability of predicate expressions. That is, one cannot write a computer program which determines validity or satisfiability.
- A predicate expression is said to be *contradicted* if it is false for all interpretations.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Models

- Let there be given a set, α, of predicate expressions,
- and an interpretation ι.
- If every w in α holds in the interpretation ι, then ι is said to be a *model* of α.

• Contexts, Interpretations and Models

- We have earlier introduced the following related terms: context and interpretation.
- It is time to sort out any possible differences in our use of the terms: model, context and interpretation.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- At the start of this section we equated, within the subject of mathematical logic, the two concepts: context and interpretation.
  - ⋆ We shall henceforth use the term context (or later the term environment) — in connection with the actual development and presentation of language interpreters — as standing for the above use of both the terms context and interpretation.
  - ⋆ And we shall, likewise, use the term interpretation to stand for the function of doing what is prescribed by such language interpreters.
  - ⋆ For matters of mathematical logic we shall not use the term context any more.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- For the term model, technical uses of this term will be in connection with the meaning of RSL definitions being sets of models: bindings between identifiers, in a space of all such, to type values (which themselves are set of values), or function values or, as we shall see, later, many other kinds of values including variables, channels.
- As another, later lecture topic we shall then discuss the looser, not necessarily technical, but usually more pragmatic use of the term model — in the senses of modelling, of creating models.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- We have introduced languages of predicate calculi. We now have several languages
  - ⋆ since we can either choose a two-valued or a three-valued logic,
  - ⋆ and since we can choose one or another set of rules of inferences.
- RSL basically has a three-valued logic.
- We say basically, as we can safely restrict particular uses of RSL to a two-valued logic — one that is consistent with a three-valued logical interpretation. That is, the **chaos** will never occur in expressions for which the two-valued logic is claimed to be sufficient.
- Whenever necessary, we are thus encouraged to state which logic we require.

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- We remind the student of the distinction between
  - ⋆ proof-theoretical (i.e., syntactical) presentations of a logic, and
  - ⋆ model-theoretical (i.e., semantical) presentations of the same logic.
- This and the previous two sections have thus provided a basis for our use of the RSL predicate calculus as a specification language.
- Since these lectures
  - ⋆ basically emphasises specification development
  - ⋆ rather than verification of such developments,
- we refer the student to specialised textbooks and monographs for more comprehensive treatments of verification.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Topic 29
Axiom Systems

- *Axioms are self-evident truths. That is, they are laws or postulates that we accept without proof.*
- In this part of the lecture we shall illustrate uses of RSL's linguistic facilities for specifying properties of sorts and functions over these sorts in terms of axioms.
- That is,
  - ⋆ in contrast to the previous three lecture parts' treatment of proof systems for logic languages, including that embedded in RSL.
  - ⋆ We shall now be using RSL itself to express axioms.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Some of the examples given now may be said to be presented prematurely or to be redundant:
  - ⋆ Either they rely on arithmetics for which no semantics, including no axioms, have been given,
  - ⋆ or they have already been presented before or will be presented more fully later.
- Be that as it may; our aim is to familiarise you with RSL specifications of axioms.

General

- An axiom system is usually a set of type definitions, a set of function signatures (of observer and generator functions, including predicates), and a set of predicate expressions (the axioms themselves).

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.53 *Euclid's Plane Geometry:* The following illustrates an axiom system. It is informally expressed:

0 Every *line* is a collection of *points*.

1 There exist at least two *points*.

2 If p and q are distinct *points*, then there exists one and only one *line* containing p and q.

3 If ℓ is a line then there exists a *point* not on ℓ.

4 If ℓ is a line and p is a point not on ℓ, then there exists one and only one *line* containing p and *parallel* to ℓ.

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- In these expressions we can identify, for example, three kinds of plane geometry terms. They are:
  - ⋆ line, point and parallel.
- We can also identify the *ontologically* determined terms:
  - ⋆ collection, containing and on;

  as well as other natural language terms.
- The axioms assume that
  - ⋆ you understand the ontologic and
  - ⋆ natural terms,

  but define, as a set of axioms,
  - ⋆ the plane geometry terms.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Axioms

- An axiom, for us, is a predicate expression that always holds, that is, which is *valid*.
- In other words, whatever quantification set is implied by some quantification range identifiers (viz. X above) they are constrained to make the axiom true.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If we, for example write:

type
  X, Y
axiom
  $\forall$ x:X $\cdot$ $\forall$ y:Y $\cdot$ x $\neq$ y

- then the sorts X and Y have at least been constrained to not contain similar elements.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- If instead

type
  X
axiom
  ∀ x:X · ∃ i:**Int** · x = i∗i

- then the sort X is the type of all square numbers.
- We could instead define X by a subtype definition:

type
  X = {| n:**Nat** · ∃ i:**Int** · n = i∗i |}

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- To repeat:
  - ⋆ Axioms are predicate expressions.
  - ⋆ Predicate expressions are only valid for certain interpretations.
  - ⋆ These interpretations are exactly what the axioms are (pragmatically) intended to model.
- Thus axioms are used to model the properties of structures, either abstract, as above, or seemingly manifest, such as the Euclidean system of plane geometry.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Axiom System

- An *axiom system*, that is, a set of predicate predicate expressions,
- also contains some type type (including sort) definitions and function signatures.
- One of the quantification range set identifiers — which may be mentioned in one or more of the axioms — are sorts, and a purpose of the axioms are to characterise those sorts.
- Usually at least one of identifiers — which may be mentioned in one or more of the axioms — is a function name, and a purpose of the axioms is to characterise that function.

Example 9.54 *Euclid's Plane Geometry:* The Euclidean geometry informally described in Example 9.53 can be formally axiomatised by first introducing the sorts P and L:

type
  P, L
value
  [0] obs_Ps: L → P-**infset**
    parallel: L × L → **Bool**

- Observe how the informal axiom in Example 9.53 has been modelled by the *observer function* obs_Ps.
- It applies to lines and yields possibly infinite sets of points.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

- Now we can introduce the axioms proper:

axiom

[1] $\exists$ p,q:P · p $\neq$ q,
[2] $\forall$ p,q:P · p $\neq$ q $\Rightarrow$
  $\exists!$ l:L · p $\in$ obs_Ps(l) $\wedge$ q $\in$ obs_Ps(l),
[3] $\forall$ l:L · $\exists$ p:P · p $\notin$ obs_Ps(l),
[4] $\forall$ l:L · $\exists$ p:P · p $\notin$ obs_Ps(l) $\Rightarrow$
  $\exists$ l′:L · l$\neq$l′ $\wedge$ p $\in$ obs_Ps(l′) $\wedge$ parallel(l,l′)

- The concept of being parallel is modelled by the predicate symbol of the same name, by its signature and by axiom [4].

■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Thus (also in RSL) an axiom system is usually represented by (i) a set of sort definitions, (ii) a set of observer and generator functions, and (iii) a set of quantified expressions, the axioms proper.

Consistency and Completeness

- A *theory* is, formally speaking, a set of axioms and a set of theorems derived, through proofs, from these axioms using the inference rules of the logic in which the axioms were stated.
- Whether the set of inference rules and the set of axioms together is sufficient for proving all valid assertions, i.e., whether the axiom system is *complete* with respect to all valid predicates, is undecidable: One cannot devise a mechanical procedure which can test an axiom system and its inference rules for completeness.

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

- Furthermore, whether the set of inference rules and the set of axioms together is such that one can prove validity of an assertion and its negation, that is, whether the axiom system is *inconsistent*, is undecidable: One cannot devise a mechanical procedure which can test an axiom systems and its inference rules for consistency.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Property-Oriented Specifications

- We give a number of examples of axiom systems.
- They each characterise one or more model(s).
- We say that they specify this (or these) model(s) in a *property-oriented* manner.
- This is as opposed to presenting the model directly in terms of for example such discrete mathematical concepts as sets, Cartesians, lists, maps, functions, etc.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.55 *Peano's Axioms:* The purpose is to define the algebra of *natural numbers* and *successor* (+1) and *equal to zero functions* (=0).

1 Zero (0) is a natural number.

2 For each natural number n there exists exactly one other natural number $n + 1$.

3 For no natural number n, is $n + 1$ equal to zero.

4 For any natural numbers m and n, if $m + 1 = n + 1$ then $m = n$.

5 For any set N of natural numbers containing zero, if $n \in A$ implies $n + 1 \in A$, then A contains every natural number.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

type N

axiom

[1] $0 \in \mathrm{N}$

[2] $\forall\ \mathrm{n{:}N} \cdot \exists!\mathrm{n'{:}N} \cdot \mathrm{n'{=}n{+}1} \wedge \mathrm{n'} \in \mathrm{N}$

[3] $\sim\exists\ \mathrm{n{:}N} \cdot \mathrm{n{+}1} = 0$

[4] $\forall\ \mathrm{m,n{:}N} \cdot \mathrm{m{+}1{=}n{+}1} \Rightarrow \mathrm{m{=}n}$

[5] $\forall\ \mathrm{A{:}N}$**-infset** $\cdot (0 \in \mathrm{A} \wedge \mathrm{n} \in \mathrm{A} \Rightarrow \mathrm{n{+}1} \in \mathrm{A}) \Rightarrow \mathrm{A} \equiv \mathrm{N}$

[5] is a specialisation of the *principle of induction:*

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

- If p is a property, i.e., p is expressible as a predicate function which may hold of (applies to) natural numbers n;
- if $p(0)$ holds;
- and if, whenever $p(n)$ holds for some natural number n, then $p(n+1)$ also holds, then that implies that all natural numbers satisfy p.
- Formulated, in general, we have:

axiom

[6] $\forall$ p:(N $\rightarrow$ **Bool**) $\cdot$ ($\forall$ n:N $\cdot$ p(n) $\Rightarrow$ p(n+1)) $\Rightarrow$ $\forall$ n:N $\cdot$ p(n)

■

Another example:

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Example 9.56 *Sine & Cosine:*

- There is given a sort of angles, A, and a sort of rational numbers, R, between −1 and 1.
- There is also given a pair of functions **sin** and **cos** (for sine, resp. cosine).
- Finally there are given the axioms:

[18] In Example 9.56 R is defined as a subtype of reals. We refer to Sect. for a proper introduction of the concept of subtypes.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

type
  A = **Real**
  R = {| r:**Real** · −1≤r≤1 |}
value
  sin,cos: A → R
axiom
  forall a:A ·
    −1 ≤ sin(a),cos(a) ≤ 1,
    $\sin^2(a) + \cos^2(a) = 1$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Here we have introduced a variant of the ∀ quantification:
  - ⋆ The keyword **forall** lets the quantifier bindings which follow it, distribute across the axioms now separated by commas.
- Under the assumption of appropriate axioms for the rational numbers, their squaring and sum, and the ≤ relation,
- Figure 9.13 exemplifies one model of this axiom.

■

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

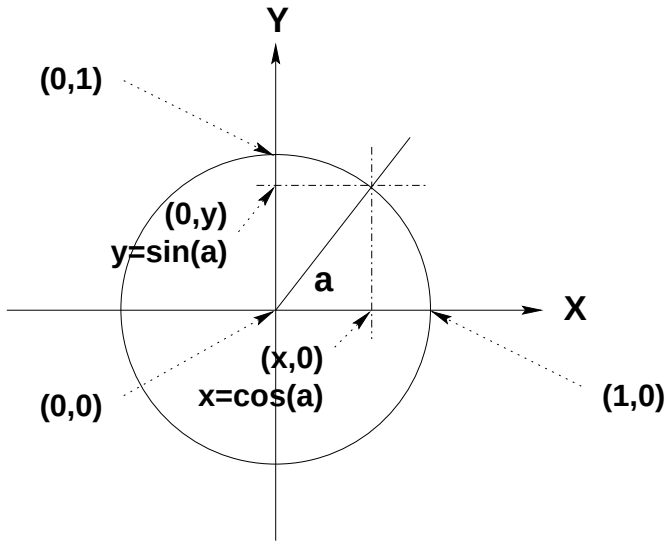

Figure 9.13: Definition of the trigonometric sin and cos functions

Further examples.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Example 9.57 *Simple Sets:*

- By a simple set we understand an unordered finite collection of simple, say in the present example, distinct atomic elements.
- Let the latter belong to sort A.
- Let the sort of simple sets be designated by S.
- Now simple sets are characterised, as already hinted at above, by being collections, by being finite, by having distinct elements, by being unordered such collections, and by the following operations:
  - ⋆ ∈ is taken as a primitive and stands for *"is the left-hand operand (an atomic element) a member of the right-hand operand (the set)."*
  - ⋆ {} is an *overloaded function symbol:*

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

  - ◇ {} either stands for the nullary constant function that yields the empty set (of no elements),
  - ◇ or {} stands for the unary function that yields the singleton set of its operand.

- ⋆ =[] stands for the unary *isempty-set* predicate function which tests whether its operand set is empty.
- ⋆ ∪ stands for the *union* operator which, when applied to two operand sets, yields the set of all elements of these operands.
- ⋆ ∩ stands for the *intersection* operator which, when applied to two operand sets, yields the set of elements common to both operands.
- ⋆ \ stands for the *set complement* operator which, when applied to two operand sets, yields the set of elements of the first operand not in second operand.
- ⋆ = stands for the *equality* operator which, when applied to two operand sets, yields truth if they are the same set, otherwise falsity.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

  - ⋆ $\subset$ stands for the *proper subset* operator which, when applied to two operand sets, yields truth if all the elements of the left-hand operand set are in the elements of the right-hand operand set and there are elements of the right-hand operand set which are not elements of the left-hand operand set.
  - ⋆ $\subseteq$ stands for the *subset* operator which, when applied to two operand sets, yields the truth if all the elements of the left-hand operand set are in the elements of the right-hand operand set.
  - ⋆ **card** stands for the *cardinality* operator which, when applied to a finite operand set, yields its number of elements.
- The axiom system provides the characterisation.
- The membership operation, $\in$, is, to repeat, taken as a primitive. That is, is not explained!

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

• A Sketch Formal Axiom System Defining S = A-set

Types and Signatures:

type
  A, S
value
  $\in, \notin$: A × S → **Bool**
  {}: **Unit** → S
  {}: A → S
  $\cup, \cap, \setminus$: S × S → S
  $=, \neq, \subset, \subseteq$: S × S → **Bool**
  card: S $\xrightarrow{\sim}$ **Nat**

Axioms:

axiom
  forall a:A, s,s′:S ·
    {a} ∈ S,

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

$$((a \in s \cup s') \equiv (a \in s \vee a \in s')),$$
$$((a \in s \cap s') \equiv (a \in s \wedge a \in s')),$$
$$((a \in s \setminus s') \equiv (a \in s \wedge a \notin s')),$$
$$s = s' \equiv (a \in s \equiv a \in s'),$$
$$s \subseteq s' \equiv (a \in s \Rightarrow a \in s'),$$
$$s \subset s' \equiv (s \subseteq s' \wedge s \neq s'),$$
$$\mathbf{card}(\{\}) \equiv 0,$$
$$a \notin s \Rightarrow \mathbf{card}(\{a\} \cup s) = 1 + \mathbf{card}(s)$$

■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

Example 9.58 *Simple Lists:*

- By a simple list we understand an ordered finite collection of, say in the present example, atomic, but not necessarily distinct elements.
- Let the latter belong to sort A.
- Let the sort of simple lists be designated by L.
- Now simple lists are characterised, as already hinted at above, by being collections, by being finite, by allowing multiple occurrence of some elements, by being ordered such collections and by the following operations:
- $\langle\rangle$, $=\langle\rangle$, **hd, tl,** $\frown$, **elems, inds, len** and $[\,\cdot\,]$.
  - $\star$ $\langle\rangle$ is an *overloaded function symbol:*

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- ⟨⟩ is (here) an *overloaded function symbol:*
  - ⟨⟩ either stands for the nullary constant function that yields the empty list (of no elements),
  - or ⟨⟩ stands for the unary function that yields the singleton list of its (only) operand.
  - =⟨⟩ stands for the unary *test for empty list predicate operator.* It applies to a list and yields truth if that list is empty, otherwise falsity.
- **hd** stands for the *head* operator which, when applied to an operand list, yields the first element of that list.
- **tl** stands for the *tail* operator which, when applied to an operand list, yields the list of all but the first element of that list, and in the same order as in the operand.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

- $\widehat{}$ stands for the *concatenation* of two operand lists of which the first must be finite. The result is the list whose first list elements are exactly those of the first operand list in the order and multiplicity of that list, and whose remaining list elements are exactly those of the last operand list in the order and multiplicity of that list.
- **elems** stands for the *elements* operator which, as a function, when applied to an operand list, yields the set of all the distinct elements of that list.
- **inds** stands for the *indices* operator which, as a function, when applied to an operand list, yields the set of all the indices into the list. If the list is of length *ell* then **inds** of that list is the set of all natural numbers from and inclusive 1 to and inclusive *ell*. If the list is empty, the yielded index set is empty.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- ⋆ **len** stands for the *length* of list operator operator which, when applied to a finite operand list, yields the length of that list, i.e., the number of not necessarily distinct elements of the list, otherwise **chaos**.
- ⋆ $\cdot(\cdot)$ stands for *list element selection*, i.e., for the (distributed fix) list operator which when applied to a "left" operand list and a "right" operand index, i.e., a natural number within the index set of the list, yields the list element having the index position in the list.

• The axiom system provides a fuller characterisation.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

• A Sketch Formal Axiom System Defining L = A*

Types and Signatures:

type
  A, L
value
  $\langle\rangle$: L
  $\langle \cdot \rangle$: A → L
  $\cdot = \langle\rangle$: L → **Bool**
  hd $\cdot$: L $\xrightarrow{\sim}$ A
  tl $\cdot$: L $\xrightarrow{\sim}$ L
  $\cdot \frown \cdot$: L × L → L
  elems $\cdot$: L → A**-set**
  inds $\cdot$: L → **Nat-set**
  le n$\cdot$: L $\xrightarrow{\sim}$ **Nat**
  $\cdot[\cdot]$: L × **Nat** $\xrightarrow{\sim}$ A

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Axioms:

axiom

$$
\begin{array}{l}
\forall\ \mathrm{a{:}A}, \ell{:}\mathrm{L} \cdot \\
\quad \langle\rangle \in \mathrm{L}, \\
\quad \langle\rangle = \langle\rangle, \\
\quad \mathbf{hd}\langle\rangle = \mathbf{chaos} \\
\quad \mathbf{hd}\langle \mathrm{a}\rangle \frown \ell \equiv \mathrm{a} \equiv (\langle \mathrm{a}\rangle \frown \ell)[\,1\,], \\
\quad \ell \frown \langle\rangle \equiv \ell \equiv \langle\rangle \frown \ell \\
\quad \mathbf{tl}\langle\rangle = \mathbf{chaos}, \\
\quad \mathbf{tl}\langle \mathrm{a}\rangle \frown \ell \equiv \ell, \\
\quad \mathbf{chaos}\,[\,\mathrm{i}\,] \equiv \mathbf{chaos},
\end{array}
$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

$$\forall\ \mathrm{i}{:}\mathbf{Nat} \cdot \mathrm{i}{>}0 \Rightarrow \mathrm{l}[\,\mathrm{i}{+}1\,] \equiv (\mathbf{tl}\ \mathrm{l})[\,\mathrm{i}\,]$$

$$\mathbf{elems}\langle\rangle \equiv \{\},\ \mathbf{elems}\langle \mathrm{a}\rangle\hat{}\mathrm{l} \equiv \{\mathrm{a}\} \cup \mathbf{elems}\ \mathrm{l}$$

$$\mathbf{inds}\langle\rangle \equiv \{\},\ \mathbf{inds}\ \mathrm{l} \equiv \{\mathrm{i}|\mathrm{i}{:}\mathbf{Nat} \cdot 1{\leq}\mathrm{i}{\leq}\mathbf{len}\ \mathrm{l}\},\ \text{i.e.,}$$

$$\mathbf{inds}\langle \mathrm{a}\rangle\hat{}\mathrm{l} \equiv \{1\} \cup \{\mathrm{i}{+}1|\mathrm{i}{:}\mathbf{Nat}{\cdot}\mathrm{i} \in \mathbf{inds}\ \mathrm{l}\}$$

$$\mathbf{len}\langle\rangle \equiv 0,\ \mathbf{len}(\langle \mathrm{a}\rangle\hat{}\mathrm{l}) \equiv 1{+}\mathbf{len}\ \mathrm{l},\ \text{i.e.,}$$

$$\mathbf{len}(\mathrm{l}\hat{}\mathrm{l}') \equiv \mathbf{len}\ \mathrm{l} + \mathbf{len}\ \mathrm{l}',$$

$$\forall\ \mathrm{i}{:}\mathbf{Nat} \cdot \mathrm{i}{>}\mathbf{len}\ \mathrm{l} \Rightarrow (\mathrm{l}\hat{}\mathrm{l}')[\,\mathrm{i}\,] \equiv \mathrm{l}'[\,\mathrm{i}{-}\mathbf{len}\ \mathrm{l}\,]$$

■

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Example 9.59 *Syntax of Simple Arithmetic Expressions:*

These examples are drawn from McCarthy.

• Analytic Syntax

- We define abstractly a small language of arithmetic expressions.
  We focus on constants, variables and infix sum and product terms.

type
  A, Term
value
  is_term: A → **Bool**
  is_const, is_var, is_sum, is_prod: Term → **Bool**
  s_addend, s_augend, s_mplier, s_mpcand: Term → Term

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11, DK-2840 Holte, Denmark
Tiergartenstraße 17, D 69121 Heidelberg, Germany

axiom

$$
\begin{array}{l}
\forall\ \text{t:Term} \cdot \\
\quad (\text{is\_const(t)} \wedge \sim(\text{is\_var(t)} \vee \text{is\_sum(t)} \vee \text{is\_prod(t)})) \wedge \\
\quad (\text{is\_var(t)} \wedge \sim(\text{is\_const(t)} \vee \text{is\_sum(t)} \vee \text{is\_prod(t)})) \wedge \\
\quad (\text{is\_sum(t)} \wedge \sim(\text{is\_const(t)} \vee \text{is\_var(t)} \vee \text{is\_prod(t)})) \wedge \\
\quad (\text{is\_prod(t)} \wedge \sim\ (\text{isc\_onst(t)} \vee \text{is\_var(t)} \vee \text{is\_sum(t)})) \wedge \\
\forall\ \text{t:A} \cdot \text{is\_term(t)} \Rightarrow \\
\quad (\text{is\_var(t)} \vee \text{is\_const(t)} \vee \text{is\_sum(t)} \vee \text{is\_prod(t)}) \wedge \\
\quad (\text{is\_sum(t)} \equiv \text{is\_term(s\_addend(t))} \wedge \text{is\_term(s\_augend(t))}) \wedge \\
\quad (\text{is\_prod(t)} \equiv \text{is\_term(s\_mplier(t))} \wedge \text{is\_term(s\_mpcand(t))})
\end{array}
$$

One could think of the following alternative, external, written representations of arithmetic expressions:

$$a + b, +ab, (\text{PLUS}\ \ A\ \ B), 7^a \times 11^b.$$

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner Fredsvej 11 DK-2840 Holte Denmark and © Springer Tiergartenstraße 17 D 69121 Heidelberg Germany

• Synthetic Syntax

A synthetic abstract syntax further introduces generators of sort values, i.e., of terms:

value

mk_sum: Term × Term → Term

mk_prod: Term × Term → Term

axiom

∀ u,v:Term •

is_sum(mk_sum(u,v)) ∧ is_prod(mk_prod(u,v)) ∧

s_addend(mk_sum(u,v)) ≡ u ∧ s_augend(mk_sum(u,v)) ≡ v ∧

s_mplier(mk_prod(u,v)) ≡ u ∧ s_mpcand(mk_prod(u,v)) ≡ v ∧

is_sum(t)⇒mk_sum(s_addend(t),s_augend(t)) ≡ t ∧

is_prod(t)⇒mk_prod(s_mplier(t),s_mpcand(t)) ≡ t

Analytic and synthetic syntaxes are truly abstract. ■

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

Discussion

- We have shown one of the most powerful means of abstraction:
  - ⋆ namely property-oriented abstraction
  - ⋆ by means of sorts,
  - ⋆ observer functions (predicates and other value "selection" functions) and
  - ⋆ generator functions.
- Specific principles of when to choose and of how to express, axiomatic property-oriented abstractions are given in later lectures.

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

Topic 30
Summary

- We have presented an overview of mathematical logic as a specification, rather than as a verification language.
- There were many parts to our exposition.
- In three stages of development we unravelled first the basis, a Boolean algebra; then a propositional logic, and finally a predicate calculus.
  - ⋆ We write an "algebra", a "logic", a "calculus", since there are many possible Boolean algebras — ours was one of a specific three-valued logic — and hence many propositional logics and predicate calculi.

Springer
Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- ⋆ We also distinguished between algebra, logic and calculus:
  - ◇ The algebra is just a simple one,
  - ◇ the logic is more extensive — and hints at a theory (with axioms, rules of inference, and theorems) which we did not elaborate on —
  - ◇ and the calculus is indeed to become a calculus: a set of rules, the inference rules, for calculation, just as the λ-calculus had rules (α-renaming and β-reduction).
- ⋆ It is the predicate calculus, for very many lectures to come, that will serve us in abstraction and in specification.

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- In our lecture on algebra we explained the notion of an algebra morphism Two algebras, one of syntax and one of semantics.
- In this chapter on logic we applied this concept repeatedly:
  - ⋆ in structuring our presentation of Boolean ground terms and their evaluation (Sect.),
  - ⋆ in structuring our presentation of propositional expressions and their evaluation,
  - ⋆ and in structuring our presentation of predicate expressions and their evaluation.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- It was perhaps not until the last of the above that we saw the full benefits of adhering to an inductive style of presenting the syntax and a homomorphic style of presenting the semantics.
- We claim that deploying the morphism idea helps structure our understanding
  - ⋆ of induction with its demand for three clauses: the basis, the inductive, and the (often implicitly understood) extremal clauses.
  - ⋆ In particular the inductive clause makes it easier for the specifier to decide on what — and how much — to develop, to define and present.
  - ⋆ Morphisms "tell" us how to develop the semantics: first the semantics corresponding to the basis clauses, then to the inductively defined syntax.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

Fredsvej 11
DK-2840 Holte
Denmark
Tiergartenstraße 17
D 69121 Heidelberg
Germany

- The choice of a three-valued logic is necessitated by our dealing,
  - ⋆ not with executable programs,
  - ⋆ but with specifications:
    - ◇ from those of abstract models of the application domain, as it is,
    - ◇ via requirements,
    - ◇ to abstract software designs.
- That choice, however, complicates the semantics and hence the proof rules.
- So far we have only presented inference rules for a two-valued logic.

Phone: +45 4221 2141, E-mail: bjorner@gmail.com, URL: www.imm.dtu.dk/~db

© Dines Bjørner and © Springer
Fredsvej 11 Tiergartenstraße 17
DK-2840 Holte D 69121 Heidelberg
Denmark Germany

- Finally, taking up a line that was begun in the lectures on algebras, we presented a thorough coverage of the predicate calculus with its quantified expressions — the practical idea of an axiom system.
  - ⋆ We applied this idea immediately, without going into logic theories of for example *undecidability* issues of axiom systems, *consistency* or *completeness*.
  - ⋆ We did so in order to present actual examples of abstract specifications.
- With a reasonable, albeit specification-oriented, view of logic, we can now proceed to apply the concepts of logic discussed in this lecture.