

---

# CS\_376 Programming with Abstract Data Types

Ulrich Berger

Autumn 2009

## The aims of the course

This course gives an introduction into the mathematical and logical foundations of formal software development.

Its main aims are

- ▶ to explain the fundamental concepts of formal logic and give you an appreciation of the power, but also the limits of formal methods,
- ▶ to make you aware of current software development tools based on formal methods,
- ▶ to enable you to apply formal techniques of verification and correct software development.

## The goal of reliable software development

The goal of formal software development is to produce reliable software, that is, computer programs that are

- (i) *adequate*: they should solve the customers problem;
- (ii) *correct*: they should be free of bugs and thus behave the way the programmer wants them to behave;
- (iii) *easy to maintain*: they should be easy to modify or extend without introducing new errors;
- (iv) *efficient* concerning time and space consumption.

# Software Crisis

It is estimated that 80% of the total time and money invested into software development is spent for finding errors and amending incorrect or poorly designed software.

Therefore, we need better programming methodologies.

## Ways out of the Software Crisis

In this course we will study reliable software development methods based on

- ▶ *mathematical modelling through Abstract Data Types*
- ▶ *formal specification*
- ▶ *formal reasoning*

## Programming support through ADTs

An *Abstract Data Type (ADT)* consists of a data structure (a collection of objects of similar “shape”) together with operations whose implementation is however *hidden*.

ADTs can be seen as small independent program units supporting

- ▶ *Modularisation*: Complex program systems are broken down into small independent and manageable parts.
- ▶ *Encapsulation*: The Implementation of an ADT is invisible from the outside and does not affect other program units.
- ▶ *Abstraction*: Unnecessary details are omitted from programs.

## Benefits of ADTs

The main advantages of programming with ADTs are:

- ▶ *Reliability*: In many cases, ADTs can be formally specified and verified, that is, the correctness of an implementation can be formally verified.
- ▶ *Safety and Maintainability*: Reimplementing an ADT (for example, for improving its efficiency) is less risky because the change does not affect other program units.
- ▶ *Flexibility*: Due to their abstractness, ADTs may often be applied to solve more programming tasks than originally intended.

## Using Mathematics and Logic to model ADTs

- ▶ *Mathematics* is used to build *models* of ADTs called *algebras*. The study of algebras, relations between algebras and mathematical operations on algebras is essential for understanding many important programming concepts.
- ▶ *Logic* is used to formally *specify* (describe) ADTs and to prove properties about them, that is, to *prove the correctness* of program units. Logic can also be used to *synthesise correct programs* automatically from a formal specification or a formal proof of a specification.
- ▶ The mathematical/logical concept of an ADT is *syntax independent* and can therefore be applied to many different programming languages.

# Course Notes and Slides

- ▶ The *course notes* contain more detailed information, examples, exercises and background on the material covered.
- ▶ Printouts of the course notes are available.
- ▶ Course notes and slides are available at the course web page <http://www.swan.ac.uk/~csulrich/cs376.html>
- ▶ The books on the next slides are recommended as further reading. They are not needed to follow the course.

## Recommended Books

[1] J Loeckx, H-D Ehrich, M Wolf, Specification of Abstract Data Types Wiley/Teubner 1996.

*Theoretical foundations of ADTs*

[2] K Meinke, J V Tucker, Universal Algebra, pp. 189-411 in Handbook of Logic in Computer Science, OUP, 1992.

*Model-theoretic aspects*

[3] A Baader, T Nipkow, Term rewriting and all that, CUP, 1998.

*Algorithmic aspects*

[4] N Dale, H M Walker, Abstract Data Types: Specification, Implementations, and Applications, D C Heath Company, 1996.

*Applications in programming*

[5] D van Dalen, Logic and Structure, 3rd edition, Springer, 1994.

*Logical foundations*

[6] A Eliens, Principles of Object-Oriented Software Development, 2nd edition, Addison Wesley 2000.

*ADTs and Object Orientation*

[7] C Okasaki, Purely Functional data Structures, CUP, 1998.

*Implementation of ADTs*

[8] D Velleman, How to Prove It, 2nd edition, CUP, 1994.

*Practical instructions of how to give a rigorous proof.*

*Many examples.*

## Coursework and Exams

- ▶ There will be *two courseworks* each worth 10%.
- ▶ The *exams* (which count 80%) will be about the material covered in the lectures and the coursework.
- ▶ Due to time restrictions, some material in the course notes may not be discussed in the lectures or the coursework (and will hence not be asked in the exams).

## Overview

- ▶ Syntax and semantics
- ▶ Formalising informal statements
- ▶ Logical Consequence, Validity, Satisfiability
- ▶ Undecidability
- ▶ Logical equivalence
- ▶ Definability
- ▶ Other Logics

# Signatures

A **many-sorted signature** is a pair  $\Sigma = (S, \Omega)$  such that

- ▶  $S$  is a nonempty set of **sorts**
- ▶  $\Omega$  is a nonempty set of **operations** of the form

$$f : s_1 \times \dots \times s_n \rightarrow s,$$

where  $s_1 \times \dots \times s_n \rightarrow s$  is called **arity** of  $f$ , with **argument sorts**  $s_1, \dots, s_n$  and **target sort**  $s$ .

Operations with zero arguments,  $c : \rightarrow s$ , are written  $c : s$  and are called **constants**. A signature must contain at least one constant  $c : s$  for each sort  $s$ .

## Algebras

A **many-sorted algebra**  $A$  for a signature  $\Sigma = (S, \Omega)$  is given by the following.

- ▶ For each sort  $s$  in  $S$  a nonempty set  $A_s$ , the **carrier set** of sort  $s$ .
- ▶ For each constant  $c: s$  in  $\Omega$  an element  $c^A \in A_s$ .
- ▶ For each operation  $f: s_1 \times \dots \times s_n \rightarrow s$  in  $\Omega$  a function

$$f^A: A_{s_1} \times \dots \times A_{s_n} \rightarrow A_s$$

## Example of a signature

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	nat, boole
<b>Constants</b>	0: nat, T: boole, F: boole
<b>Operations</b>	add: nat $\times$ nat $\rightarrow$ nat $\leq$ : nat $\times$ nat $\rightarrow$ boole

Example of a  $\Sigma$ -algebra

<b>Algebra</b>	$A$
<b>Carriers</b>	$\mathbf{N}, \mathbf{B}$
<b>Constants</b>	$0, \mathbf{T}, \mathbf{F}$
<b>Operations</b>	$+: \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ $\leq: \mathbf{N} \times \mathbf{N} \rightarrow \mathbf{B}$

This means:

$$A_{\text{nat}} = \mathbf{N} := \{0, 1, 2, \dots\} \quad (\text{natural numbers})$$

$$A_{\text{boole}} = \mathbf{B} := \{\mathbf{T}, \mathbf{F}\} \quad (\text{boolean values})$$

$$0^A = 0, \quad \mathbf{T}^A = \mathbf{T}, \quad \mathbf{F}^A = \mathbf{F}$$

$$\text{add}^A = + \quad (\text{addition on natural numbers})$$

$$\leq^A = \leq \quad (\text{the usual less-or-equal relation on natural numbers})$$

## Another algebra for the signature $\Sigma = (\text{nat}, \text{boole}, 0, \text{T}, \text{F}, \text{add}, \leq)$

<b>Algebra</b>	$B$
<b>Carriers</b>	$\mathbf{N}^+, \mathbf{B}$
<b>Constants</b>	$1, \text{T}, \text{F}$
<b>Operations</b>	$*$ : $\mathbf{N}^+ \times \mathbf{N}^+ \rightarrow \mathbf{N}^+$ $ $ : $\mathbf{N}^+ \times \mathbf{N}^+ \rightarrow \mathbf{B}$

This means:

$$B_{\text{nat}} = \mathbf{N}^+ := \{1, 2, \dots\} \quad (\text{positive natural numbers})$$

$$B_{\text{boole}} = \mathbf{B}$$

$$0^B = 1, \quad \text{T}^B = \text{T}, \quad \text{F}^B = \text{F}$$

$$\text{add}^B = * \quad (\text{multiplication})$$

$$\leq^B = | \quad (\text{divisibility})$$

## Yet another algebra for the signature $\Sigma = (\text{nat}, \text{boole}, 0, \text{T}, \text{F}, \text{add}, \leq)$

<b>Algebra</b>	$C$
<b>Carriers</b>	$\{0\}, \{0\}$
<b>Constants</b>	$0, 0, 0$
<b>Operations</b>	$f_0: \{0\} \times \{0\} \rightarrow \{0\}$ $f_0: \{0\} \times \{0\} \rightarrow \{0\}$

This means:

$$C_{\text{nat}} = \{0\}$$

$$C_{\text{boole}} = \{0\}$$

$$0^C = 0, \quad \text{T}^C = 0, \quad \text{F}^C = 0$$

$$\text{add}^C = f_0 \quad (\text{the only function from } \{0\} \times \{0\} \text{ to } \{0\})$$

$$\leq^C = f_0$$

# Terms

Let  $\Sigma = (S, \Omega)$  be a signature. We assume that for every sort  $s \in S$  a set  $X_s$  of **variables** is given.

**$\Sigma$ -terms** and their sorts are defined by the following rules.

- (i) Every variable  $x \in X_s$  is a term of sort  $s$ .
- (ii) Every constant  $c$  in  $\Sigma$  of sort  $s$  is a term of sort  $s$ .
- (iii) If  $f: s_1 \times \dots \times s_n \rightarrow s$  is an operation in  $\Sigma$ , and  $t_1, \dots, t_n$  are terms of sorts  $s_1, \dots, s_n$ , respectively, then

$$f(t_1, \dots, t_n)$$

is a term of sort  $s$ .

## Some terminology for terms

- ▶ The set of all terms of sort  $s$  with variables in  $X$  is denoted by  $T(\Sigma, X)_s$ .
- ▶ A term is **closed** if it doesn't contain variables, i.e. is built without the use of rule (i).
- ▶ The set of all closed terms of sort  $s$  is denoted by  $T(\Sigma)_s$ .
- ▶ Clearly  $T(\Sigma)_s = T(\Sigma, \emptyset)_s$ .

## Examples of terms for the signature $\Sigma = (\text{nat}, \text{boole}, 0, \text{T}, \text{F}, \text{add}, \leq)$

$x$

$0$

$\text{add}(0, y)$

$\text{add}(\text{T}, \text{F})$

$\text{add}(\text{add}(0, x), y, z)$

$\text{add}(\text{add}(0, 0), \text{add}(x, x))$

$\text{add}(0, \text{add}(0, \text{add}(0, 0)))$

Which of these are wellformed  $\Sigma$ -terms?

Which of the terms are closed?

## Semantics of terms

Let  $A$  be an algebra for the signature  $\Sigma = (S, \Omega)$ , and let  $X = (X_s)_{s \in S}$  a set of variables.

A **variable assignment**  $\alpha: X \rightarrow A$  is a function assigning to every variable  $x \in X_s$  an element  $\alpha(x) \in A_s$ .

Given a variable assignment  $\alpha: X \rightarrow A$  we define for each term  $t \in \mathbb{T}(\Sigma, X)_s$  its **value**

$$t^{A, \alpha} \in A_s$$

- (i)  $x^{A, \alpha} := \alpha(x)$ .
- (ii)  $c^{A, \alpha} := c^A$ .
- (iii)  $f(t_1, \dots, t_n)^{A, \alpha} := f^A(t_1^{A, \alpha}, \dots, t_n^{A, \alpha})$ .

For closed terms  $t$ , i.e.  $t \in \mathbb{T}(\Sigma)$ , we write  $t^A$  instead of  $t^{A, \alpha}$ .

## Example and exercise

$$\alpha: \{x, y\} \rightarrow \mathbf{N},$$

$$\alpha(x) := 3 \text{ and } \alpha(y) := 5.$$

Compute the values of the terms shown earlier in the algebra  $A$ .

## Formulas

We fix again a signature  $\Sigma$  and a set  $X_s$  of variables for every sort  $s$  in  $\Sigma$ .

- (i)  $\perp$  is a formula, called *absurdity*.
- (ii) If  $t_1, t_2 \in T(\Sigma, X)$  are terms of the same sort then  $t_1 = t_2$  is a formula.

- (iii) If  $P$  and  $Q$  are formulas then so are

$$P \rightarrow Q, \quad P \wedge Q, \quad P \vee Q$$

- (iv) If  $P$  is a formula then so are

$$\forall x P, \quad \exists x P$$

for every variable  $x \in X$ .

## Free variables

- ▶ A **free occurrence** of a variable  $x$  in a formula  $P$  is an occurrence of  $x$  in  $P$  which is not in the scope of a quantifier  $\forall x$  or  $\exists x$ .
- ▶  $FV(P) :=$  the set of free variables of  $P$ .
- ▶  $\mathcal{L}(\Sigma, X) := \{P \mid P \text{ } \Sigma\text{-formula, } FV(P) \subseteq X\}$
- ▶  $\mathcal{L}(\Sigma) := \mathcal{L}(\Sigma, \emptyset)$ ,
- ▶ A formula  $P$  is closed if  $FV(P) = \emptyset$ .
- ▶  $\mathcal{L}(\Sigma)$  is the set of closed  $\Sigma$ -formulas (that is,  $\mathcal{L}(\Sigma) = \mathcal{L}(\Sigma, \emptyset)$ ).

## Remarks

- ▶ Formulas as defined above are usually called **first-order formulas**, since we allow quantification over object variables only. If we would also quantify over set variables we would obtain second-order formulas.
- ▶ A formula is **quantifier free**, **qf** for short, if it doesn't contain quantifiers.
- ▶ An **equation** is a formula of the form  $t_1 = t_2$ .
- ▶ A formula is **universal** if it is of the form  $\forall x_1 \dots \forall x_n P$  where  $P$  is quantifier free.

## Abbreviations

Formula

Abbreviation

 $P \rightarrow \perp$  $\neg P$  (negation) $(P \rightarrow Q) \wedge (Q \rightarrow P)$  $P \leftrightarrow Q$  (equivalence) $t = \text{T}$  $t$ 

where

 $t$  is of sort boole

Formula

Abbreviation

$$\forall x_1 \forall x_2 \dots \forall x_n P$$

$$\forall x_1, x_2, \dots, x_n P$$

$$\exists x_1 \exists x_2 \dots \exists x_n P$$

$$\exists x_1, x_2, \dots, x_n P$$

$$\forall x_1, \dots, x_n P,$$

where

$$\{x_1, \dots, x_n\} = \text{FV}(P)$$

$$\forall P \quad (\text{closure of } P)$$

## Examples and exercises

$$P_1 \quad :\equiv \quad F = F$$

$$P_2 \quad :\equiv \quad x = 0 \rightarrow \text{add}(y, x) = x$$

$$P_3 \quad :\equiv \quad \exists y (x = y \rightarrow \forall z x = z)$$

$$P_4 \quad :\equiv \quad \forall x (\leq (y, x) = \text{T})$$

$P_4$  can be abbreviated  $\forall x (y \leq x)$ .

- Compute the free variables of these formulas.
- Which formulas are closed?
- Which formulas are quantifier free?
- Which formulas are universal?
- Which formulas are equations?

## Semantics of formulas

Let

- ▶  $A$  a  $\Sigma$ -algebra,
- ▶  $P$  a  $\Sigma$ -formula,
- ▶  $\alpha$  a **variable assignment**, that is, a mapping assigning to every variable  $x$  of sort  $s$  an element  $\alpha(x) \in A_s$ .

We define the relation

$$A, \alpha \models P$$

which can be equivalently read as

*$P$  holds in  $A$  under the assignment  $\alpha$*

or  *$A, \alpha$  is a model of  $P$*

## Semantics of atomic formulas

- (i)  $A, \alpha \not\models \perp$ , i.e.  $A, \alpha \models \perp$  does not hold.
- (ii)  $A, \alpha \models t_1 = t_2$  iff  $t_1^{A, \alpha} = t_2^{A, \alpha}$ .

## Semantics of the propositional connectives

(iii)  $A, \alpha \models P \wedge Q$  iff  $A, \alpha \models P$  and  $A, \alpha \models Q$ .

$A, \alpha \models P \vee Q$  iff  $A, \alpha \models P$  or  $A, \alpha \models Q$ .

$A, \alpha \models P \rightarrow Q$  iff  $A, \alpha \models P$  implies  $A, \alpha \models Q$   
(that is  $A, \alpha \not\models P$  or  $A, \alpha \models Q$ ).

These definition correspond to the following truth tables:

$P$	$Q$	$P \wedge Q$	$P \vee Q$	$P \rightarrow Q$
T	T	T	T	T
T	F	F	T	F
F	T	F	T	T
F	F	F	F	T

## Semantics of quantified formulas

- (iv)  $A, \alpha \models \forall x P$  iff  $A, \alpha_x^a \models P$  for all  $a \in A_s$   
 $A, \alpha \models \exists x P$  iff  $A, \alpha_x^a \models P$  for at least one  $a \in A_s$

Here, it is assumed that the variable  $x$  is of sort  $s$ .

The modified variable assignment  $\alpha_x^a$  is defined as follows:

$$\alpha_x^a(x) := a$$

$$\alpha_x^a(y) := \alpha(y) \text{ if } y \text{ is different from } x$$

## Formalisation of informal statements

Consider the following statement (Velleman, p. 65):

Everyone has a relative he doesn't like.

In order to formalise this statement we need to determine:

- (1) the signature,
- (2) the formula (representing the statement).

Exercise:

1. Carry out the formalisation of the statement above.
2. Choose an algebra and decide whether the statement holds in it.

## More formalisation exercises

Formalise the following statements:

- (a) Not every real number has a square root.
- (b) Every positive real number has a square root.
- (c) The cube of a real number is bigger than its square.
- (d) Mary saw a seal and John saw it too.
- (e) Mary saw a seal and John saw a seal too.
- (f) A shark doesn't sleep.
- (g) A shark was spotted in Swansea Bay.

For each statement choose (the intended) algebra and decide whether the statement holds in the chosen algebra.

## Tarski's World

Exercise: Formalise the following statements and evaluate them in Socrates' world.

1.  $f$  is the largest block (all others are smaller).
2. All dodecs are of different sizes.
3. All blocks behind  $f$  are cubes.
4. All cubes are behind a tet.
5. For every tet there is a larger cube.
6. Tets are smaller than cubes.
7. no block is smaller than all others.
8. All dodecs are isolated.
9. There is a block between a tet and a cube.
10. There is exactly one largest block and it is a cube.
11. Every block has exactly one shape.

## Mathematical and logical truth

Let  $A$  be a  $\Sigma$ -algebra,  $\alpha_0: \emptyset \rightarrow A$  the empty variable assignment and  $P$  a closed  $\Sigma$ -formula.

### Mathematical truth

$$A \models P \quad : \Leftrightarrow \quad A, \alpha_0 \models P$$

(“ $P$  is true in  $A$ ”; “ $P$  holds in  $A$ ”)

If the algebra  $A$  is the “intended mathematical structure” (for example the algebra of the natural numbers or the real numbers), then one simply says “ $P$  is mathematically true”.

### Logical truth

$$\models P \quad : \Leftrightarrow \quad A \models P \text{ for all } \Sigma\text{-algebras } A$$

(“ $P$  is logically true”, “ $P$  is valid”, “ $P$  is a logical tautology” )

## Models of an axiom system

Let  $\Gamma$  be an “axiom system”, that is, a set of **closed**  $\Sigma$ -formulas.

### Model

$$A \models \Gamma \quad : \Leftrightarrow \quad A \models P \text{ for all } P \in \Gamma$$

(“ $A$  is a model of  $\Gamma$ ”)

## Logical consequence

$$\Gamma \models P \quad : \Leftrightarrow \quad P \text{ is true in all models of } \Gamma$$

that is,

$$\Gamma \models P \quad : \Leftrightarrow \quad \text{for all } A \ (A \models \Gamma \Rightarrow A \models P)$$

(“ $P$  is a **logical consequence** of  $\Gamma$ ”; “ $\Gamma$  **logically implies**  $P$ ”)

Note that  $\emptyset \models P \Leftrightarrow \models P$

## Satisfiability

A set of closed  $\Sigma$ -formulas  $\Gamma$  is said to be **satisfiable** if it has a model, that is, there exists a  $\Sigma$ -algebra  $A$  in which all formulas of  $\Gamma$  are true ( $A \models \Gamma$ ).

Validity and satisfiability are related by the following equivalences:

$$\begin{aligned} P \text{ valid} &\Leftrightarrow \{\neg P\} \text{ unsatisfiable (that is, not satisfiable)} \\ P \text{ satisfiable} &\Leftrightarrow \{\neg P\} \text{ not valid} \end{aligned}$$

## Undecidability of Logical Truth

### **Theorem (A Church)**

It is undecidable whether or not a formula is logically true.

(Proof: reduce the halting problem to the validity problem (i.e. code Turing machines into logic))

However: There is an effective procedure generating all valid formulas (technically: the set of valid formulas is recursively enumerable).

⇒ next chapter.

# Undecidability of Mathematical Truth

## Theorem (K Gödel, A Tarski)

It is undecidable whether or not a formula in the language of arithmetic ( $0, 1, +, *, \leq$ ) is mathematically true, that is, holds in the standard algebra of natural numbers.

Moreover, the set of mathematically true formulas is **not** recursively enumerable, that is, there is no effective procedure that produces all true mathematical formulas.

## Examples

We work with a signature with the sorts `nat` and `boole` and the operation `<`: `nat × nat → boole`.

$$\exists x \forall y (x < y) \rightarrow \forall y \exists x (x < y)$$

is a tautology (logically true).

$$\forall x, y (x < y \rightarrow \exists z (x < z \wedge z < y))$$

is satisfiable, but not a tautology (why?).

## Examples

$$\Gamma := \{\forall x \neg(x < x), \forall x, y, z (x < y \wedge y < z \rightarrow x < z)\}$$

The formula

$$P := \forall x, y (x < y \rightarrow \neg y < x)$$

is a logical consequence of  $\Gamma$ , that is,

$$\Gamma \models P$$

.

## Logical equivalence

Two formulas  $P$  and  $Q$  are called *logically equivalent* if the closed formula

$$\forall(P \leftrightarrow Q)$$

is logically valid.

Since logically equivalent formulas have the same truth value in each algebra they are freely interchangeable as part of another formula:

If  $P$  and  $Q$  are logically equivalent, so are  $R[P]$  and  $R[Q]$ .

Therefore, logical equivalences are useful to bring a formula in a simpler yet logically equivalent form.

On the next few slides we list some important logical equivalences.

## Examples of logical equivalences: propositional formulas

$$P \quad \neg\neg P$$

$$P \rightarrow Q \quad \neg Q \rightarrow \neg P$$

$$P \rightarrow Q \quad \neg P \vee Q$$

$$\neg(P \rightarrow Q) \quad P \wedge \neg Q$$

$$\neg(P \wedge Q) \quad \neg P \vee \neg Q$$

$$\neg(P \vee Q) \quad \neg P \wedge \neg Q$$

$$P \wedge (Q \vee R) \quad (P \wedge Q) \vee (P \wedge R)$$

$$(P \vee Q) \rightarrow R \quad (P \rightarrow Q) \wedge (Q \rightarrow R)$$

## Proving equivalences

In order to prove that  $P$  and  $Q$  are equivalent one has to prove that the formulas  $P \rightarrow Q$  and  $Q \rightarrow P$  are both logically valid.

In order to prove that  $P \rightarrow Q$  is logically valid, one assumes that  $P$  holds and shows that then  $Q$  holds as well.

## Examples of logical equivalences: quantifier formulas

$$\neg \forall x P(x) \qquad \exists x \neg P(x)$$

$$\neg \exists x P(x) \qquad \forall x \neg P(x)$$

$$\forall x (P(x) \wedge Q(x)) \qquad \forall x P(x) \wedge \forall x Q(x)$$

$$\exists x (P(x) \vee Q(x)) \qquad \exists x P(x) \vee \exists x Q(x)$$

$$\exists x P(x) \rightarrow Q \qquad \forall x (P(x) \rightarrow Q) \quad (x \notin \text{FV}(Q))$$

Exercise: Prove these equivalences.

## Examples of non-equivalences

Show that the following pairs of formulas are *not* logical equivalences:

$$\forall x (P(x) \vee Q(x)) \quad \forall x P(x) \vee \forall x Q(x)$$

$$\exists x (P(x) \wedge Q(x)) \quad \exists x P(x) \wedge \exists x Q(x)$$

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

However, in each of these pairs one half of the equivalence holds.  
Which?

## Definability

Let  $A$  be a  $\Sigma$ -algebra and  $s_1, \dots, s_n$  sorts in  $\Sigma$ .

A subset  $X \subseteq A_{s_1} \times \dots \times A_{s_1}$  is **definable**

(or  **$\Sigma$ -definable**, or **first-order definable**)

if there is a  $\Sigma$ -formula  $P$  with  $FV(P) \subseteq \{x_1, \dots, x_n\}$  such that for all  $\vec{a} \in A_{s_1} \times \dots \times A_{s_1}$

$$\vec{a} \in X \iff A \models P(\vec{a})$$

where  $A \models P(\vec{a})$  is shorthand for  $A, \alpha_{\vec{x}}^{\vec{a}} \models P$ .

## Exercise on definability

Consider the signature  $\Sigma$  with the sorts `boole`, `nat`, the constants  $T, F: \text{boole}$  and  $0, 1: \text{nat}$  as well as the operations  $+, *: \text{nat} \times \text{nat} \rightarrow \text{nat}$ .

Let  $A$  be the  $\Sigma$ -algebra of natural numbers with the expected interpretation of  $\Sigma$ .

Show that the following sets and relations are definable:

- ▶ The set of odd numbers
- ▶ The relation  $<$
- ▶ The divisibility relation
- ▶ The set of prime numbers

# First-order Logic

First-order predicate logic is a **general purpose logic**. It is used to

- ▶ formulate and answer questions concerning the foundations of mathematics,
- ▶ specify and verify programs.

## Other Logics

Some specialized sub-logics tailored for specific kinds of problems in computer science:

- ▶ Hoare logic – imperative programming
- ▶ higher-order logic – functional programming
- ▶ clausal logic – logic programming, AI
- ▶ modal/temporal/process logic – distributed processes
- ▶ equational logic – hardware, rapid prototyping
- ▶ bounded/linear logic – complexity analysis
- ...

## Exercise: the value of a term

Compute the value of the term

$$t := f(x, f(x, x))$$

in the algebra  $C$  with carrier  $\mathbf{N}$  and

$$f^C(n, m) := n + 2 * m$$

under the valuation  $\alpha$  with  $\alpha(x) := 3$ .

## Exercise: logical equivalence

Show that the formula

$$\forall x \exists y (y \neq x \wedge P(y))$$

and the formula

$$\exists x \exists y (y \neq x \wedge P(x) \wedge P(y))$$

are logically equivalent.

Hint: for one direction one needs that the carrier sets of algebras are always non-empty.

## Exercise: formalisation

Let  $\Sigma$  be the signature with one sort, one constant,  $0$ , and two binary operations,  $+$  and  $*$ .

Let  $R$  be the  $\Sigma$ -algebra with the real numbers as carrier, the constant  $0$ , and the usual addition and multiplication of real numbers.

Write down  $\Sigma$ -formulas that express in  $R$  the following statements:

- (a)  $x \leq y$
- (b)  $x = 1$ .
- (c)  $x = \sqrt{2}$ .

In other words, show that the relation  $\{(x, y) \in R \mid x \leq y\}$  and the sets  $\{1\}$  and  $\{\sqrt{2}\}$  are definable.

Generalising (b): Show that every rational number is definable.

## Exercise: quantifier elimination

Let  $\Sigma$  be the signature with the sorts *real* and *boole*, one constant, 0, two binary operations, + and \* and a binary boolean operation <.

Let  $R$  be the  $\Sigma$ -algebra with the real numbers and booleans as carriers, the constant 0, and the usual addition, multiplication and order relation of real numbers.

Find a *quantifier free*  $\Sigma$ -formula  $P(a, b, c, d)$  such that

$$R \models \forall a, b, c, d (\exists x (a * x^3 + b * x^2 + c * x + d = 0) \leftrightarrow P(a, b, c, d))$$

where  $x^3$  and  $x^2$  are shorthand for  $x * x * x$  and  $x * x$ , respectively.

Hint: Do a case analysis on whether  $a = 0$  and use your school knowledge on quadratic equations.

# Proofs

- ▶ Natural Deduction
- ▶ Equality rules
- ▶ Soundness and Completeness
- ▶ Axioms and rules for data types

We will now study a **proof calculus** for deriving facts of the form

$$\Gamma \models P$$

One of the most celebrated results in logic is

### **Gödel's Completeness Theorem.**

It states that there are a few simple **proof rules** that suffice to derive **all** true statements of the form  $\Gamma \models P$ .

In particular all tautologies can be derived (set  $\Gamma := \emptyset$ ).



Kurt Gödel (1906-1978)

## Natural Deduction

In the following we study **Natural Deduction**, a proof calculus introduced by Gentzen and further developed by Prawitz.

Other important (and equivalent) proof calculi are the **Sequent Calculus** and the **Hilbert Calculus**.

We have chosen Natural Deduction because

- (1) it is close to the natural way of human reasoning and thus easy to learn;
- (2) it is closely related to functional programming and thus particularly suited for program synthesis from proofs.

## The rules of natural deduction

For each logical connective and quantifier there are

**Introduction Rules** describing how to **obtain** a formula

**Elimination Rules** describing how to **use** a formula

constructed by that connective or quantifier.

## Conjunction

	Introduction	Elimination	
$\wedge$	$\frac{P \quad Q}{P \wedge Q} \wedge^+$	$\frac{P \wedge Q}{P} \wedge_l^-$	$\frac{P \wedge Q}{Q} \wedge_r^-$

## Example

A derivation of  $Q \wedge P$  from the (free) assumption  $P \wedge Q$ :

$$\frac{\frac{P \wedge Q}{Q} \wedge_r^- \quad \frac{P \wedge Q}{P} \wedge_l^-}{Q \wedge P} \wedge^+$$

From this example we see that a derivation (= formal proof) is a tree labelled by formulas and rule names (e.g.  $\wedge^+$ ,  $\wedge_l^-$ ,  $\wedge_r^-$ ).

The leaves of the rules are the assumptions of the derivation, the root is the conclusion.

The proof rules determine whether or not a derivation is correct.

The correctness of a derivation can be easily checked (by a program).

## Exercise

Derive from the assumption

$$P \wedge (Q \wedge R)$$

the formula

$$(P \wedge Q) \wedge R.$$

## Implication

	Introduction	Elimination
$\rightarrow$	$\frac{u : P \quad \vdots \quad Q}{P \rightarrow Q} \rightarrow^+ u$	$\frac{P \rightarrow Q \quad P}{Q} \rightarrow^-$

In the rule  $\rightarrow^+$  the assumption  $u : P$  is *discharged*.

The rule  $\rightarrow^-$  is also called *Modus Ponens*.

## Example

A derivation of  $P \wedge Q \rightarrow Q \wedge P$  (without free assumptions).

$$\frac{\frac{\frac{u : P \wedge Q}{Q} \wedge_r^- \quad \frac{\frac{u : P \wedge Q}{P} \wedge_l^-}{Q \wedge P} \wedge^+}{P \wedge Q \rightarrow Q \wedge P} \rightarrow^+ u}$$

## Example

A derivation of  $P \rightarrow (Q \rightarrow R)$  from the free assumption  $P \wedge Q \rightarrow R$ .

$$\frac{\frac{\frac{P \wedge Q \rightarrow R}{P \wedge Q} \rightarrow^- \quad \frac{\frac{u : P \quad v : Q}{P \wedge Q} \wedge^+}{R} \rightarrow^+ v}{Q \rightarrow R} \rightarrow^+ v}{P \rightarrow (Q \rightarrow R)} \rightarrow^+ u$$

The free (or un-discharged) assumptions are those assumptions at the leaves of a proof tree that have not been discharged by a  $\rightarrow^+$  rule (and hence are **not** labelled by “ $u :$ ” or “ $u :$ ”).

## Example

$$\frac{\frac{u : P}{Q \rightarrow P} \rightarrow^+ v}{P \rightarrow (Q \rightarrow P)} \rightarrow^+ u$$

This example shows that assumptions given by the use of an implication introduction rule do not have to be used: in the derivation above the assumption  $v : Q$  is not used.

## Proof strategy

**Construct derivations “bottom up”**

**Exercise:** Derive from the assumption  $P \rightarrow (Q \rightarrow R)$  the formula  $P \wedge Q \rightarrow R$ .

A slightly different notation for  $\rightarrow^+$ 

Sometimes derivations become more readable when in the implication introduction rule the label is written  $\rightarrow^+ u : P$  instead of just  $\rightarrow^+ u$ . A previous example derivations then reads:

$$\frac{P \wedge Q \rightarrow R \quad \frac{\frac{u : P \quad v : Q}{P \wedge Q} \wedge^+}{\rightarrow^-} \quad \frac{R}{Q \rightarrow R} \rightarrow^+ v : Q}{P \rightarrow (Q \rightarrow R)} \rightarrow^+ u : P$$

## Making the free assumptions explicit

Sometimes it is useful to state explicitly the available assumptions. Here are the proof rules for  $\wedge$  and  $\rightarrow$  written in this style:

	Introduction	Elimination	
$\wedge$	$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \wedge^+$	$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash P} \wedge_l^-$	$\frac{\Gamma \vdash P \wedge Q}{\Gamma \vdash Q} \wedge_r^-$
$\rightarrow$	$\frac{\Gamma, P \vdash Q}{\Gamma \vdash P \rightarrow Q} \rightarrow^+$	$\frac{\Gamma \vdash P \rightarrow Q \quad \Gamma \vdash P}{\Gamma \vdash Q} \rightarrow^-$	

In the  $\rightarrow$ -introduction rule,  $\Gamma, P$  is shorthand for  $\Gamma \cup \{P\}$ .

## The assumption rule

Writing derivations with explicit assumptions the use of a free assumption has the form of a rule without premises:

Assumption	$\Gamma, P \vdash P$
------------	----------------------

## Example of a derivation with explicit assumptions

$$\frac{
 \frac{
 \frac{
 \frac{
 \frac{
 P \wedge Q \rightarrow R, P, Q \vdash P \wedge Q \rightarrow R
 }{
 P \wedge Q \rightarrow R, P, Q \vdash P
 }
 \rightarrow^-
 }{
 P \wedge Q \rightarrow R, P, Q \vdash P \wedge Q
 }
 \wedge^+
 }{
 P \wedge Q \rightarrow R, P, Q \vdash R
 }
 }{
 P \wedge Q \rightarrow R, P, Q \vdash R
 }
 }{
 P \wedge Q \rightarrow R, P \vdash Q \rightarrow R
 }
 \rightarrow^+
 }{
 P \wedge Q \rightarrow R \vdash P \rightarrow (Q \rightarrow R)
 }
 \rightarrow^+$$

One sees that this style, although very concise, is impractical when it comes to writing down concrete derivations manually.

## Notation

$$P_1 \rightarrow P_2 \rightarrow \dots \rightarrow P_{n-1} \rightarrow P_n$$

abbreviates

$$P_1 \rightarrow (P_2 \rightarrow \dots \rightarrow (P_{n-1} \rightarrow P_n) \dots)$$

**Exercise:**

Derive  $(P \rightarrow Q \rightarrow R) \rightarrow (P \rightarrow Q) \rightarrow P \rightarrow R$ .

## Disjunction

	Introduction	Elimination
$\vee$	$\frac{P}{P \vee Q} \vee_l^+$ $\frac{Q}{P \vee Q} \vee_r^+$	$\frac{P \vee Q \quad P \rightarrow R \quad Q \rightarrow R}{R} \vee^-$

The rule  $\vee^-$  represents *proof by cases*.

## Example

Deriving  $Q \vee P$  from  $P \vee Q$ .

$$\frac{P \vee Q \quad \frac{\frac{u : P}{Q \vee P} \vee_r^+ \quad P \rightarrow Q \vee P}{\rightarrow^+ u : P} \quad \frac{\frac{v : Q}{Q \vee P} \vee_l^+ \quad Q \rightarrow Q \vee P}{\rightarrow^+ v : Q} \vee^-}{Q \vee P}$$

## Proof strategy

Working bottom up

- ▶ use **disjunction elimination** (with  $P \vee Q$  say) as **early** as possible, as it reduces a goal  $R$  to the simpler goals  $P \rightarrow R$  and  $Q \rightarrow R$ ;
- ▶ use **disjunction introduction** as **late** as possible: Proving  $P \vee Q$  by disjunction introduction means to commit yourself to proving either  $P$  or  $Q$ , so the goal  $P \vee Q$  is replaced by the harder goal  $P$  (or  $Q$ ).

## Exercise

Prove

$$(P \wedge Q) \vee (P \wedge R) \rightarrow P \wedge (Q \vee R).$$

## Falsity

	Introduction	Elimination	
$\perp$	No rule	$\frac{\perp}{P}$ efq	$\frac{\neg\neg P}{P}$ raa

**efq = ex-falso-quodlibet:**

From falsity deduce anything.

**raa = reductio-ad-absurdum:**

To prove  $P$ , it suffices to reduce  $\neg P$  to absurdity (= falsity), that is, prove  $\neg P \rightarrow \perp$  ( $\equiv \neg\neg P$ ).

## Example: Peirce's law

$$\begin{array}{c}
 \frac{u : \neg P}{\frac{\frac{\frac{\perp}{\neg\neg P} \rightarrow^+ u : \neg P}{P} \text{raa}}{\rightarrow^+ w : (P \rightarrow Q) \rightarrow P}} \rightarrow^- \\
 \frac{w : (P \rightarrow Q) \rightarrow P}{\frac{u : \neg P}{P} \rightarrow^-} \rightarrow^- \\
 \frac{\frac{\frac{\frac{u : \neg P \quad v : P}{\perp} \text{efq}}{P \rightarrow Q} \rightarrow^+ v : P}{\rightarrow^-}}{\rightarrow^-}
 \end{array}$$

## raa implies efq

The rule efq is weaker than raa in the sense that the former can be obtained from the latter:

From the assumption  $\perp$  we can derive any formula  $P$  without using efq, but using raa instead:

$$\frac{\frac{\perp}{(P \rightarrow \perp) \rightarrow \perp} \rightarrow^+ u : P \rightarrow \perp}{P} \text{raa}$$

However, proofs avoiding raa (but using efq only) are particularly interesting because they represent constructive reasoning and correspond directly to programs (see third part of the course notes).

# Universal Quantification

Introduction	Elimination
$\frac{P(x)}{\forall x P(x)} \forall^+ \quad (*)$	$\frac{\forall x P(x)}{P(t)} \forall^-$

(\*) provided  $x$  does not occur free in any free assumption above that rule.

# Universal Quantification: rules with explicit assumptions

Introduction	Elimination
$\frac{\Gamma \vdash P(x)}{\Gamma \vdash \forall x P(x)} \forall^+ \quad (*)$	$\frac{\Gamma \vdash \forall x P(x)}{\Gamma \vdash P(t)} \forall^-$

(\*) provided  $x$  does not occur free in  $\Gamma$ .

## Example

$$\frac{\frac{\forall x P(x)}{P(x+1)} \forall^-}{\forall x P(x+1)} \forall^+$$

The variable condition is satisfied, because  $x$  is not free in  $\forall x P(x)$ .

## Incorrect derivations

What is wrong with the following 'derivations'?

$$\frac{\frac{\forall y(x < 1 + y)}{x < 1 + 0} \forall^-}{\forall x(x < 1 + 0)} \forall^+$$

$$\frac{\frac{\forall x(\forall y(x < y + 1) \rightarrow x = 0)}{\forall y(y < y + 1) \rightarrow y = 0} \forall^-}{\frac{y = 0}{\forall y(y = 0)} \forall^+} \forall y(y < y + 1) \rightarrow -$$

## Exercise

Prove

$$\forall x (P(x) \rightarrow Q(x)) \rightarrow \forall x P(x) \rightarrow \forall x Q(x)$$

## Existential Quantification

Introduction	Elimination
$\frac{P(t)}{\exists x P(x)} \exists^+$	$\frac{\exists x P(x) \quad \forall x (P(x) \rightarrow Q)}{Q} \exists^- \quad (*)$

(\*) provided  $x$  is not free in  $Q$

## Example

$$\frac{
 \frac{
 \frac{
 u : \forall x (x - 1) + 1 = x
 }{(x - 1) + 1 = x} \forall^-
 }{\exists y (y + 1 = x)} \exists^+
 }{\forall x \exists y (y + 1 = x)} \forall^+
 }{\forall x ((x - 1) + 1 = x) \rightarrow \forall x \exists y (y + 1 = x)} \rightarrow^+ u$$

## Exercise

Derive from  $\forall x(P(x) \rightarrow Q(f(x)))$  the formula

$$\exists x P(x) \rightarrow \exists y Q(y)$$

## Equality rules

**Reflexivity**  $\frac{}{t = t}$  refl

**Symmetry**  $\frac{s = t}{t = s}$  sym

**Transitivity**  $\frac{r = s \quad s = t}{r = t}$  trans

**Compatibility**  $\frac{s_1 = t_1 \quad \dots \quad s_n = t_n}{f(s_1, \dots, s_n) = f(t_1, \dots, t_n)}$  comp

## Example

$$\frac{\frac{\frac{\forall x \forall y (x + y = y + x)}{\forall y (0 + y = y + 0)} \forall^-}{0 + x = x + 0} \forall^-}{0 + x = x} \forall^- \quad \frac{\frac{\forall x (x + 0 = x)}{x + 0 = x} \forall^-}{\text{trans}} \quad \frac{}{y = y} \text{ refl}$$


---


$$\frac{\frac{(0 + x) * y = x * y}{\forall y ((0 + x) * y = x * y)} \forall^+}{\forall x \forall y ((0 + x) * y = x * y)} \forall^+ \quad \text{comp}$$

## Classical, intuitionistic, and minimal derivability

$\Gamma \vdash_c P$  :  $\Leftrightarrow \Gamma \vdash d: P$  for some derivation  $d$ .  
( $P$  is derivable from  $\Gamma$  in **classical logic**)

$\Gamma \vdash_i P$  :  $\Leftrightarrow \Gamma \vdash d: P$  for some derivation  $d$  not using  
the rule reductio-ad-absurdum.  
( $P$  is derivable from  $\Gamma$  in **intuitionistic logic**)

$\Gamma \vdash_m P$  :  $\Leftrightarrow \Gamma \vdash d: P$  for some derivation  $d$  using  
neither the rule reductio-ad-absurdum nor the rule  
ex-falso-quodlibet.  
( $P$  is derivable from  $\Gamma$  in **minimal logic**)

## Fundamental properties of substitution

### Lemma

Let  $t(x)$  be a term possibly containing the variable  $x$ , and let  $r, s$  be terms of the same sort as  $x$ . Then

$$r = s \vdash_m t(r) = t(s)$$

**Proof.** Induction on  $t(x)$ .

### Lemma

Let  $P(x)$  be a formula possibly containing the variable  $x$ , and let  $r, s$  be terms of the same sort as  $x$ . Then

$$r = s \vdash_m P(r) \leftrightarrow P(s)$$

**Proof.** Induction on the formula  $P(x)$ .

# Soundness

## Soundness Theorem

If  $\Gamma \vdash_c P$  then  $\Gamma \models P$ .

**Proof.** The theorem follows immediately from the following statement which can be easily shown by induction on derivations:

For every finite set of (not necessarily closed) formulas  $\Gamma$  and every formula  $P$ ,

if  $\Gamma \vdash_c P$  then  $A, \alpha \models P$  for all algebras  $A$  and variable assignments  $\alpha$  such that  $A, \alpha \models \Gamma$

# Completeness

## Completeness Theorem (Gödel)

If  $\Gamma \models P$  then  $\Gamma \vdash_c P$ .

The proof of this theorem is beyond the scope of this course.

## Consistency and Satisfiability

A (possibly infinite) set of formulas  $\Gamma$  is called **consistent** if  $\Gamma \not\vdash_c \perp$ , that is, there is no (classical) derivation of  $\perp$  from assumptions in  $\Gamma$ .

In other words: A set of formulas  $\Gamma$  is consistent if and only if no contradiction can be derived from  $\Gamma$ .

### Satisfiability Theorem

Every consistent set of formulas has a model.

**Exercise.** Prove the Satisfiability Theorem from the Completeness Theorem and vice versa.

## Compactness

### Compactness Theorem

Let  $\Gamma$  be an axiom system such that every finite subset of  $\Gamma$  has a model.

Then  $\Gamma$  has a model.

**Proof.** By the Satisfiability Theorem it suffices to prove that  $\Gamma$  is consistent.

Assume not. Then  $\Gamma \vdash_c \perp$ , that is, there exists a derivation of  $\perp$  from assumptions in  $\Gamma$ .

Since the derivation is finite it can have used as assumptions only finitely many formulas in  $\Gamma$ . In other words  $\Gamma_0 \vdash_c \perp$  for some **finite** subset  $\Gamma_0$  of  $\Gamma$ . Hence  $\Gamma_0$  has no model (by the Soundness Theorem), contradicting the assumption.

## Application of compactness: Infinitesimals

### Theorem

There exists an algebra  ${}^*\mathbf{R}$  that satisfies the same formulas as the algebra  $\mathbf{R}$  of real numbers, but such that there exists an *infinitesimal*, that is, a positive number  $\delta \in {}^*\mathbf{R}$  which is "infinitely small" in the sense that

$$0 < \delta < \frac{1}{n}$$

for every natural number  $n > 0$ .

This result is the basis of **Nonstandard Analysis** (introduced by A. Robinson in a different way). In Nonstandard Analysis one can interpret expressions like  $\frac{df(x)}{dx}$  as  $\frac{f(x+\delta)-f(x)}{\delta}$  where  $\delta$  is an infinitesimal, thus making the naive intuition of differentiation precise.

## Infinitesimals ctd.

### Proof of the existence of an algebra ${}^*\mathbf{R}$ with infinitesimals.

Let  $\Gamma$  be the set of closed formulas that hold in  $\mathbf{R}$ . Let  $c$  be a new constant and consider the set of closed formulas

$$\Delta := \{0 < c\} \cup \{n \cdot c < 1 \mid n \in \mathbf{N}\}$$

where  $n \cdot c$  is shorthand for the term  $c + \dots + c$  ( $n$ -times  $c$ ).

Then every finite subset of  $\Gamma \cup \Delta$  has a model.

Therefore, by the Compactness Theorem,  $\Gamma \cup \Delta$  has a model  ${}^*\mathbf{R}$ .

The algebra  ${}^*\mathbf{R}$  satisfies all formulas that hold in  $\mathbf{R}$ , and in addition the number  $\delta := c$  is an infinitesimal.

## Axioms for the booleans

**Boole 1**      $\overline{T \neq F}$  boole1

**Boole 2**      $\overline{\forall x (x = T \vee x = F)}$  boole2

**Lemma**

We can derive  $\forall x (\neg x \leftrightarrow x = F)$  without assumptions.

**Proof:** Exercise

## Peano Axioms

$$\text{Peano 1} \quad \frac{}{0 \neq t + 1} \text{peano1}$$

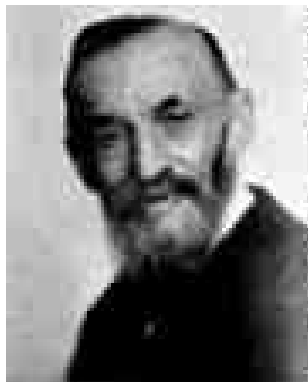
$$\text{Peano 2} \quad \frac{}{s + 1 = t + 1 \rightarrow s = t} \text{peano2}$$

$$\text{Induction} \quad \frac{P(0) \quad \forall x (P(x) \rightarrow P(x + 1))}{\forall x P(x)} \text{ind}$$

### Remarks

1. In applications there will be further axioms describing additional operations on the booleans and natural numbers. Examples are, the equations defining addition by primitive recursion from zero and the successor function.

2. Similar axioms can be introduced for data types such as lists or trees.



Giuseppe Peano (1858-1932)

## Other proof systems

Important proof systems other than Natural Deduction are

- ▶ Sequent Calculus
- ▶ Hilbert systems
- ▶ Resolution

They have different advantages and disadvantages and are used for different purposes.

In addition to first-order logic they can be applied to other logics such as modal logic, temporal logic, type theory, etc.

The following slides briefly recap the characteristics of natural deduction and also sketch the characteristics of the other proof systems.

For simplicity, we only look at propositional rules (no quantifiers).

## Natural Deduction

**Characteristics:** Introduction and elimination rules.

**Advantages:** Close to natural human reasoning; easy to use.

**Usage:** Manual theorem proving, interactive theorem proving, intuitionistic type theory.

**Relation to programming:** Closely related to functional programming. This can be exploited for the automatic extraction of correct programs from proofs.

**Implementation:** Agda, Isabelle, Coq, Minlog.

## Sequent Calculus

**Characteristics:** Left and Right introduction rules (below is one example), Cut rule.

$\wedge$	$\frac{\Gamma, P, Q \vdash R}{\Gamma, P \wedge Q \vdash R} \wedge\text{Left}$	$\frac{\Gamma \vdash P \quad \Gamma \vdash Q}{\Gamma \vdash P \wedge Q} \wedge\text{Right}$
Cut	$\frac{\Gamma \vdash P \quad \Gamma, P \vdash Q}{\Gamma \vdash Q} \text{Cut}$	

**Usage:** Proof search, analysis of proofs (proof theory).

## Hilbert Systems

**Characteristics:** Only one rule (modus ponens, that is  $\rightarrow$ -elimination), many axioms, no binding of assumptions.

Examples of axioms:  $P \wedge Q \rightarrow P$ ,  $P \wedge Q \rightarrow Q$ ,  $P \rightarrow Q \rightarrow P \wedge Q$ ,  $P \rightarrow P \vee Q$ ,  $Q \rightarrow P \vee Q$ ,  $(P \rightarrow R) \rightarrow (Q \rightarrow R) \rightarrow (P \vee Q \rightarrow R)$ , etc.

**Advantages:** Easy to implement (because of absence of assumption binding).

**Usage:** Automated theorem proving.

**Relation to programming:** Corresponds to a combinatorial style of programming.

## Resolution

**Characteristics:** For formulas in *conjunctive normal* form only:  
 $C_1 \wedge \dots \wedge C_n$  where each  $C_i$  is a *clause* of the form  $L_1 \vee \dots \vee L_k$   
 and each  $L_j$  is a *literal*, that is, an atomic formula or the negation  
 of an atomic formula.

There is only one proof rule: Resolution.

$$\frac{\dots \wedge (L_1 \vee \dots \vee P \vee \dots \vee L_k) \wedge (L'_1 \vee \dots \vee \neg P \vee \dots \vee L'_m) \wedge \dots}{\dots \wedge (L_1 \vee \dots \vee L_k \vee L'_1 \vee \dots \vee L'_m) \wedge \dots}$$

**Usage:** Automated theorem proving, SAT solving.

**Implementation:** SAT solvers, for example OK-solver.

## Summary

- ▶ Derivations: minimal, intuitionistic and classical.
- ▶ Soundness and Completeness Theorem for First-Order Logic.
- ▶ Axioms and rules for equality and data types.

## Exercises: Minimal logic

(a)  $(P \rightarrow \neg Q) \rightarrow (Q \rightarrow \neg P)$

(b)  $(P \rightarrow (Q \rightarrow R)) \rightarrow ((P \rightarrow Q) \rightarrow (P \rightarrow R))$

(c)  $P \rightarrow \neg\neg P$

(d)  $\neg(P \wedge \neg P)$

(e)  $(P \wedge (Q \vee R)) \leftrightarrow ((P \wedge Q) \vee (P \wedge R))$

(f)  $(P \vee Q) \rightarrow \neg(\neg P \wedge \neg Q)$

(g)  $\neg(P \leftrightarrow \neg P)$

## Exercises: Intuitionistic logic

(a)  $(P \wedge \neg P) \rightarrow R$

(b)  $(\neg P \vee Q) \rightarrow (P \rightarrow Q)$

(c)  $(\neg\neg P \rightarrow P) \leftrightarrow ((\neg P \rightarrow P) \rightarrow P)$

(d)  $(P \vee Q) \rightarrow (\neg P \rightarrow Q)$

## Exercises: Classical logic

(a)  $\neg\neg P \rightarrow P$

(b)  $(\neg P \rightarrow P) \rightarrow P$

(c)  $P \vee \neg P$

(d)  $\neg(\neg P \wedge \neg Q) \rightarrow (P \vee Q)$

(e)  $\neg(\neg P \vee \neg Q) \rightarrow (P \wedge Q)$

(f)  $\neg(P \rightarrow Q) \rightarrow P \wedge \neg Q$

## Exercises: Minimal logic

$$(a) \quad \forall x (P(x) \wedge Q(x)) \leftrightarrow (\forall x P(x) \wedge \forall x Q(x))$$

$$(b) \quad \exists x (P(x) \vee Q(x)) \leftrightarrow (\exists x P(x) \vee \exists x Q(x))$$

$$(c) \quad (\forall x P(x) \vee \forall x Q(x)) \rightarrow \forall x (P(x) \vee Q(x))$$

$$(d) \quad \exists x (P(x) \wedge Q(x)) \rightarrow (\exists x P(x) \wedge \exists x Q(x))$$

## Exercises: Classical logic

$$(a) \exists x \neg P(x) \leftrightarrow \neg \forall x P(x)$$

$$(b) \neg \exists x P(x) \leftrightarrow \forall x \neg P(x)$$

$$(c) \exists x (R(x) \rightarrow \forall y R(y))$$

Remark: In part (c), if one interprets the formula  $R(x)$  as “it is raining on day  $x$ ”, then the formula  $\exists x (R(x) \rightarrow \forall y R(y))$  can be read as a weatherforecast that is guaranteed to be valid:

“There exists a day such that, if it is raining on that day, then it will be raining every day”

## Exercise: Completeness

Let  $\Gamma$  be a **complete axiom system**, that is, a set of closed formulas such that for every closed formula  $P$  either  $\Gamma \models P$  or  $\Gamma \models \neg P$ .

Let  $A$  be a model of  $\Gamma$ .

Show that  $A \models P$  iff and only if  $\Gamma \models P$ , for every closed formula  $P$ .

## Exercise: Decidability

An example of a complete axiom system are the axioms of a *dense linear order without endpoints*, i.e. the universal closures of

$$x \leq x, \quad x \leq y \wedge y \leq z \rightarrow x \leq z, \quad x \leq y \wedge y \leq x \rightarrow x = y,$$

$$x \leq y \vee y \leq x, \quad x < z \rightarrow \exists y (x < y \wedge y < x),$$

$$\exists y (x < y), \quad \exists y (y < x).$$

where  $x < y$  abbreviates  $x \leq y \wedge x \neq y$ . The rational numbers are a model of this axiom system.

Use these facts and the result of the previous exercise to show that that the truth of arbitrary statements about order relations between rational numbers can be effectively decided.

# Abstract Data Types

- ▶ Homomorphisms and abstract data types
- ▶ Reducts, Subalgebras and Quotients
- ▶ The Homomorphism Theorem
- ▶ Summary and Exercises

### ADTs in programming books/courses:

- ▶ One or more data types together with a bunch of operations.
- ▶ Implementation details of the data types and the operations are hidden.
- ▶ Makes software more **modular** and thus **robust** against changes of implementation details.

### Mathematical modelling of ADTs:

- ▶ A class of algebras that is closed under **isomorphisms**.
- ▶ Each member of the class represents a particular implementation of the ADT.
- ▶ Mathematical modelling is **syntax free** and thus independent of the choice of programming language.

## Homomorphisms

Given:

A signature  $\Sigma = (S, \Omega)$ ,

$\Sigma$ -algebras  $A, B$ .

A **Homomorphism**,  $\varphi: A \rightarrow B$  is a family of mappings

$$\varphi = (\varphi_s)_{s \in S}, \quad \varphi_s: A_s \rightarrow B_s$$

such that

- ▶  $\varphi_s(c^A) = c^B$
- ▶  $\varphi_s(f^A(a_1, \dots, a_n)) = f^B(\varphi_{s_1}(a_1), \dots, \varphi_{s_n}(a_n))$

Short notation for the second equation:

$$\varphi_s \circ f^A = f^B \circ (\varphi_{s_1}, \dots, \varphi_{s_n})$$

## Special Homomorphisms

<i>Name</i>	<i>Property</i>
monomorphism	injective
epimorphism	surjective
isomorphism	bijjective
endomorphism	$\varphi: A \rightarrow A$
automorphism	bijjective endomorphism

## Example of an Isomorphism

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	$R$
<b>Constants</b>	$0: \text{nat}$
<b>Operations</b>	$\text{add}: R \times R \rightarrow R$

<b>Algebra</b>	$A$
<b>Carriers</b>	$\mathbf{R}$ (real numbers)
<b>Constants</b>	$0$
<b>Operations</b>	$+: \mathbf{R} \times \mathbf{R} \rightarrow \mathbf{R}$

<b>Algebra</b>	$B$
<b>Carriers</b>	$\mathbf{R}^+ := \{x \in \mathbf{R} \mid x > 0\}$
<b>Constants</b>	$1$
<b>Operations</b>	$*: \mathbf{R}^+ \times \mathbf{R}^+ \rightarrow \mathbf{R}^+$

$$\varphi: \mathbf{R} \rightarrow \mathbf{R}^+, \quad \varphi(x) := 2^x$$

## Example of a Homomorphism

### *Exercises*

1. Verify that  $\varphi: \mathbf{R} \rightarrow \mathbf{R}^+$ ,  $\varphi(x) = 2^x$  is indeed an isomorphism between the algebras  $A := (\mathbf{R}, 0, +)$  and  $B := (\mathbf{R}^+, 1, *)$ .
2. Consider the algebra  $B'$  that is defined like  $B$ , but where 0 is added to the carrier set (the constant is still interpreted as 1 and the operation as multiplication).

Show that there exists no epimorphism from  $A$  to  $B'$ .

Conclude that  $A$  and  $B'$  are not isomorphic.

## Composition of homomorphisms

Given:

$\Sigma = (S, \Omega)$  signature

$A, B, C$   $\Sigma$ -algebras.

Let  $\varphi: A \rightarrow B$ ,  $\psi: B \rightarrow C$  be homomorphisms.

Define  $\psi \circ \varphi := (\psi_s \circ \varphi_s)_{s \in S}$ .

### Theorem

Homomorphisms are closed under composition, that is,

if  $\varphi: A \rightarrow B$  and  $\psi: B \rightarrow C$  are homomorphisms,  
then  $\psi \circ \varphi: A \rightarrow C$  is a homomorphism.

## Inverse of an isomorphism

Let  $\varphi: A \rightarrow B$  be an isomorphism.

Define  $\varphi^{-1} := (\varphi_s^{-1})_{s \in S}$ .

### Theorem

Isomorphisms are closed under inverses, that is,

if  $\varphi: A \rightarrow B$  is an isomorphism,  
then  $\varphi^{-1}: B \rightarrow A$  is an isomorphism.

The equivalence relation “ $A \simeq B$ ”

$A \simeq B \quad :\Leftrightarrow \quad$  there exists an isomorphism from  $A$  to  $B$   
 (“ $A$  and  $B$  are isomorphic”)

$\text{Alg}(\Sigma) :=$  the class of all  $\Sigma$ -algebras.

**Theorem**

“ $A \simeq B$ ” is an equivalence relation on  $\text{Alg}(\Sigma)$ , that is, for all  $\Sigma$ -algebras  $A, B, C$  we have

- (i) *Reflexivity:*  $A \simeq A$ ,
- (ii) *Symmetry:* if  $A \simeq B$  then  $B \simeq A$ ,
- (ii) *Transitivity:*  
if  $A \simeq B$  and  $B \simeq C$  then  $A \simeq C$ .

# Abstract Data Types

An **abstract data type (ADT)** for a signature  $\Sigma$  is a class  $\mathcal{C} \subseteq \text{Alg}(\Sigma)$  of  $\Sigma$ -algebras which is closed under isomorphisms, that is,

$$\text{if } A \in \mathcal{C} \text{ and } A \simeq B \text{ then } B \in \mathcal{C}.$$

An ADT  $\mathcal{C}$  is called **monomorphic** if all its elements are isomorphic, that is,

$$\text{if } A \in \mathcal{C} \text{ and } B \in \mathcal{C} \text{ then } A \simeq B.$$

Otherwise  $\mathcal{C}$  is called **polymorphic**.

## Examples of ADTs

- (a)  $\text{Alg}(\Sigma)$  is a polymorphic ADT.
- (b) For each  $\Sigma$ -algebra  $A$  the class  $\{B \in \text{Alg}(\Sigma) \mid B \simeq A\}$  is a monomorphic ADT.
- (c) The class of finite  $\Sigma$ -algebras,  $\{A \in \text{Alg}(\Sigma) \mid \text{all } A_s \text{ finite}\}$ , is a polymorphic ADT.

**Exercise** Is  $\mathcal{C} := \{A \in \text{Alg}(\Sigma) \mid \text{all } A_s = \mathbf{N}\}$  an ADT?

(d) Let  $\Sigma$  a one sorted signature with one constant, 1, and one binary operation,  $*$ . A *monoid* is a  $\Sigma$ -algebra  $M$  satisfying the following equations (for all  $x, y, z \in M$ ):

$$1 * x = x$$

$$x * 1 = x$$

$$x * (y * z) = (x * y) * z$$

The class  $\mathcal{MON}$  of all monoids is a polymorphic ADT.

- ▶ How do we prove that  $\mathcal{MON}$  is an ADT?
- ▶ How do we prove that  $\mathcal{MON}$  is polymorphic?

**Answer:** Using formal specification ( $\Rightarrow$  next chapter).

More interesting examples of ADTs will be given later when we study specifications of ADTs.

## Subsignatures

Let  $\Sigma = (S, \Omega)$ ,  $\Sigma' = (S', \Omega')$  be signatures.

$\Sigma$  is a **subsignature** of  $\Sigma'$ , written  $\Sigma \subseteq \Sigma'$ , if

$$S \subseteq S' \text{ and } \Omega \subseteq \Omega'$$

In that case we also say  $\Sigma'$  is an **expansion** of  $\Sigma$ .

## Reducts and Expansions

Let  $\Sigma$  be a subsignature of  $\Sigma'$ .

To every  $\Sigma'$ -algebra  $A$  we can construct a  $\Sigma$ -algebra  $B$  by 'throwing away' all parts of  $A$  not named in  $\Sigma$ , i.e.

- ▶  $B_s := A_s$  for all sorts  $s$  in  $\Sigma$ ,
- ▶  $c^B := c^A$  for all constants  $c$  in  $\Sigma$ ,
- ▶  $f^B := f^A$  for all operations  $f$  in  $\Sigma$ ,

We call  $B$  the  $\Sigma$ -**reduct** of  $A$  and denote it by  $A|_{\Sigma}$ .

We also say that  $A$  is an **expansion** of  $B$ .

Given an ADT  $\mathcal{C}$  for a signature  $\Sigma'$  and a subsignature  $\Sigma$  of  $\Sigma'$  we define the  $\Sigma$ -reduct of  $\mathcal{C}$  by  $\mathcal{C}|_{\Sigma} := \{A|_{\Sigma} \mid A \in \mathcal{C}\}$ .

It is easy to see that  $\mathcal{C}|_{\Sigma}$  is an ADT.

## Subalgebras

Let  $A$  and  $B$  be algebras for the *same* signature  $\Sigma = (S, \Omega)$ .

$A$  is a **subalgebra** of  $B$  if

$$A_s \subseteq B_s$$

$$c^A = c^B$$

$$f^A(a_1, \dots, a_n) = f^B(a_1, \dots, a_n) \quad \text{for all } a_i \in A.$$

We also say that  $B$  is an **extension**  $A$ .

Since every algebra  $A$  can be viewed as a monomorphic ADT (the class of algebras isomorphic to  $A$ ), the relations “reduct”, “expansion”, “subalgebra”, “extension” can also be viewed as relations between monomorphic ADTs. In other words, we understand these relations “modulo isomorphism”. For example, in this way  $A$  is a subalgebra of  $B$  if  $A$  is isomorphic to some algebra  $A'$  such that  $A'$  is a subalgebra of  $B$ .

## Example: Four algebras and their relationships

- ▶  $\mathbf{N}_{+*}$  := Natural numbers with 0, addition and multiplication.
- ▶  $\mathbf{N}_+$  := Natural numbers with 0 and addition.
- ▶  $\mathbf{E}_{+*}$  := Even natural numbers with 0, addition and multiplication
- ▶  $\mathbf{E}_+$  := Even natural numbers with 0 and addition.
- ▶ What are their subalgebra/reduct relationships?

$\mathbf{N}_+$  ?  $\mathbf{N}_{+*}$

$\mathbf{N}_+$  is a reduct of  $\mathbf{N}_{+*}$ .

$\mathbf{E}_{+*}$  ?  $\mathbf{N}_{+*}$

$\mathbf{E}_{+*}$  is a subalgebra of  $\mathbf{N}_{+*}$ .

$\mathbf{E}_+$  ?  $\mathbf{N}_+$

$\mathbf{E}_+$  is a subalgebra of  $\mathbf{N}_+$ .

$\mathbf{E}_+$  ?  $\mathbf{E}_{+*}$

$\mathbf{E}_+$  is a reduct of  $\mathbf{E}_{+*}$ .

$\mathbf{E}_+$  subalgebra of  $\mathbf{N}_{+*}$ ?

No, signatures are not the same.

$\mathbf{E}_+$  reduct of  $\mathbf{N}_{+*}$ ?

No, carriers are not the same.

## Homomorphic image

Let  $\varphi: A \rightarrow B$  be a homomorphism.

The **homomorphic image** of  $A$  under  $\varphi$  is defined as

$$\varphi(A_s) := \{\varphi_s(a) \mid a \in A_s\} \subseteq B_s$$

$$\varphi(A) := (\varphi(A_s))_{s \in S}$$

The homomorphic image is a subalgebra of  $B$ .

Proof: Exercise.

## Congruences and quotients

A **congruence** on a  $\Sigma$ -algebra  $A$  is a family

$$\sim = (\sim_s)_{s \in S}$$

of equivalence relations on  $A_s$  such that

$$a_i \sim_{s_i} b_i \quad (1 \leq i \leq n) \quad \Rightarrow \\ f^A(a_1, \dots, a_n) \sim_s f^A(b_1, \dots, b_n)$$

for  $a_i \in A$ .

The **congruence class** of an element  $a \in A_s$  is

$$[a]_{\sim_s} := \{b \in A_s \mid a \sim_s b\}$$

## Quotient algebra

The **Quotient algebra**  $A/\sim$  is defined as follows:

$$(A/\sim)_s := \{[a]_{\sim_s} \mid a \in A_s\}$$

$$c^{A/\sim} := [c^A]_{\sim_s}$$

$$f^{A/\sim}([a_1]_{\sim_{s_1}}, \dots, [a_n]_{\sim_{s_n}}) := [f^A(a_1, \dots, a_n)]_{\sim_s}$$

Exercise: Prove that the operations  $f^{A/\sim}$  are well-defined.

## Example

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	nat, list
<b>Constants</b>	0, 1: nat, empty: list
<b>Operations</b>	add: nat $\times$ nat $\rightarrow$ nat cons: nat $\times$ list $\rightarrow$ list @: list $\times$ list $\rightarrow$ list length: list $\rightarrow$ nat

<b>Algebra</b>	$A$
<b>Carriers</b>	$\mathbf{N}, \mathbf{N}^*$
<b>Constants</b>	0, 1, nil
<b>Operations</b>	+ : $\mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ cons : $\mathbf{N} \times \mathbf{N}^* \rightarrow \mathbf{N}^*$ @ : $\mathbf{N}^* \times \mathbf{N}^* \rightarrow \mathbf{N}^*$ length : $\mathbf{N}^* \rightarrow \mathbf{N}$

$$n \sim_{\text{nat}} m \quad : \Leftrightarrow \quad n = m$$

$$s \sim_{\text{list}} t \quad : \Leftrightarrow \quad \forall n: s \text{ contains } n \text{ if and only if } t \text{ contains } n$$

- ▶ Is  $\sim := (\sim_{\text{nat}}, \sim_{\text{list}})$  a congruence on  $A$ ?

Let  $B$  be the reduct of  $A$  obtained by removing the length function.

- ▶ Is  $\sim$  a congruence on  $B$ ?
- ▶ If yes, what is the quotient algebra?  $B / \sim$ ?

## The Homomorphism Theorem

Let  $\varphi: A \rightarrow B$  be a homomorphism. Define

$$a \sim_{\varphi,s} b \quad :\Leftrightarrow \quad \varphi_s(a) = \varphi_s(b)$$

Then  $\sim_{\varphi} := (\sim_{\varphi,s})_{s \in S}$  is a congruence on  $A$ , and

$$A/\sim_{\varphi} \quad \simeq \quad \varphi(A)$$

with canonical isomorphism

$$[\varphi]: A/\sim_{\varphi} \rightarrow \varphi(A)$$

$$[\varphi]_s([a]_{\sim_{\varphi,s}}) := \varphi_s(a)$$

## What does all this mean to the programmer?

- ▶ An ADT corresponds to a *programming task*.
- ▶ Fulfilling that task means to *implement* an ADT, that is, to implement an algebra in the ADT.
- ▶ Improving an implementation means to replace one implementation of an ADT by another (better) one.
- ▶ Implementing or improving software without being given an ADT is meaningless (though common practice).
- ▶ The operations *reduct*, *expansion*, *subalgebra*, *extension*, *quotient* are frequently used to construct new from old (implementations of) ADTs.
- ▶ Formal specifications (using logical formulas) can be used to accurately describe ADTs ( $\Rightarrow$  next chapter).

- ▶ The *Homomorphism Theorem* is a standard tool for obtaining an implementation of an ADT from a given implementation (of another ADT):
  - Suppose  $A$  is the (concrete) implementation of an ADT  $\mathcal{A}$  and  $B$  is an element of an ADT  $\mathcal{B}$  which is given mathematically (no implementation yet). We wish to have an implementation of  $\mathcal{B}$ .
  - This can be accomplished by an epimorphism  $\varphi: A \rightarrow B$ :
  - $A/\sim_\varphi$  is an implementation of  $\mathcal{B}$  because  $A/\sim_\varphi \simeq \varphi(A) = B$  according to the homomorphism theorem.

## Example: Implementing the Rational Numbers

- ▶ Assume we have an implementation  $Z$  of the ADT of *integers* with  $0$ ,  $1$ ,  $+$ ,  $-$  and  $*$ .
- ▶ We wish to obtain an implementation of the ADT  $\mathbf{Q}$  of *rational numbers* with  $0$ ,  $1$ ,  $+$ ,  $-$ ,  $*$ ,  $/$  and an undefined element (to cater for division by  $0$ ).

## Implementing the rationals (ctd.)

**Step 1:** Extend  $Z$  to an algebra  $A_1$  which has as objects pairs of integers  $(n, m)$  (intended to represent  $n/m$ ) and constants and operations defined as follows:

$$0^{A_1} := (0, 1)$$

$$1^{A_1} := (1, 1)$$

$$(n_1, m_1) +^{A_1} (n_2, m_2) := (n_1 * m_2 + n_2 * m_1, m_1 * m_2)$$

$$(n_1, m_1) -^{A_1} (n_2, m_2) := (n_1 * m_2 - n_2 * m_1, m_1 * m_2)$$

$$(n_1, m_1) *^{A_1} (n_2, m_2) := (n_1 * n_2, m_1 * m_2)$$

where  $+$ ,  $-$ ,  $*$  denote the implementations of addition, subtraction and multiplication in  $Z$ .

Exercise: Show that  $Z$  is a subalgebra of  $A_1$  by defining a monomorphism from  $Z$  to  $A_1$  and applying the Homomorphism Theorem.

## Implementing the rationals (ctd.)

**Step 2:** Form a *subalgebra*  $A_2$  of  $A_1$  by removing all elements of the form  $(n, 0)$ .

Exercise: Show that  $A_2$  is indeed a subalgebra of  $A_1$ .

**Step 3:** *Extend*  $A_2$  by an element  $u$  for “undefined” to obtain  $A_3$ , extending the operations in a **strict** way, that is, the result is  $u$  if one of the arguments is  $u$ .

**Step 4:** *Expand*  $A_3$  to a  $A_4$  by adding to the signature a binary operation  $/$  and defining

$$\begin{aligned}(n_1, m_1) /^{A_4} (n_2, m_2) &:= (n_1 * m_2, m_1 * n_2) \quad \text{if } n_2 \neq 0 \\ &:= u \quad \text{if } n_2 = 0\end{aligned}$$

Furthermore,  $/^{A_4}$  shall be strict, that is,  $x /^{A_4} y := u$  if  $x = u$  or  $y = u$ .

## Implementing the rationals (ctd.)

**Step 5:** Define an *epimorphism*  $\varphi : A_4 \rightarrow \mathbf{Q}$  by

$$\varphi(n, m) := n/m, \quad \varphi(u) := u$$

Exercise: Check that  $\varphi$  is a homomorphism!

**Step 6:** Set  $Q := A_4 / \sim_\varphi$ . According to the *Homomorphism Theorem*,  $Q$  is isomorphic to  $\mathbf{Q}$  and therefore an implementation of  $\mathbf{Q}$ .

## Generators

Let  $\Sigma = (S, \Omega)$  be a signature and  $\Omega' \subseteq \Omega$  a subset of constants and operations. Set  $\Sigma' := (S, \Omega')$ .

A  $\Sigma$ -algebra  $A$  is called **generated** by  $\Omega'$  if every  $a \in A_s$  is the value of a closed  $\Sigma'$ -term, i.e. for every  $a \in A_s$  there exists a term  $t \in \mathbb{T}(\Sigma')$  such that  $t^A = a$ .

In that case  $\Omega'$  is called a **set of generators** of  $A$ .

If  $\Omega' = \Omega$ , then one simply says that  $A$  is generated.

$A$  is called **freely generated** by  $\Omega'$  if every  $a \in A_s$  is the **unique** value of a closed  $\Sigma'$ -term, i.e. for every  $a \in A_s$  there exists a **unique** term  $t \in \mathbb{T}(\Sigma')$  such that  $t^A = a$ .

## Minimal generators

A **minimal set of generators** of an algebra  $A$  is a set of generators  $\Omega'$  such that no proper subset of  $\Omega'$  generates  $A$ .

It is easy to see that a set of free generators is always minimal. The converse does not hold in general.

## Examples

1. The closed term algebra  $\mathbb{T}(\Sigma)$  is freely generated, since for every closed term  $\Sigma$ -term  $t$  we have  $t^{\mathbb{T}(\Sigma)} = t$ .
2. The Algebra of natural numbers with 0 and addition is not generated, since 0 is the only natural number which is the value of a closed term.
3. If we add to the algebra in example 2 the successor operation `succ` (that adds 1 to a number), we obtain an expansion that is generated. This expansion is not freely generated because there are natural numbers that can be defined in different ways. For example,  $0 = 0 + 0$ .  
However, there exists a subset of generators that freely generate the algebra, namely 0 and `succ`. This is so, because natural numbers can be **uniquely** written in the form `succ(...succ(0)...)...`.  
0 and `succ` also form a minimal set of generators.

## Exercise

Discuss the following algebra concerning generators:

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	nat, list
<b>Constants</b>	0, 1: nat, empty: list
<b>Operations</b>	add: nat $\times$ nat $\rightarrow$ nat cons: nat $\times$ list $\rightarrow$ list @ : list $\times$ list $\rightarrow$ list length: list $\rightarrow$ nat

<b>Algebra</b>	$A$
<b>Carriers</b>	$\mathbf{N}$ , $\mathbf{N}^*$
<b>Constants</b>	0, 1, nil
<b>Operations</b>	+ : $\mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ cons : $\mathbf{N} \times \mathbf{N}^* \rightarrow \mathbf{N}^*$ @ : $\mathbf{N}^* \times \mathbf{N}^* \rightarrow \mathbf{N}^*$ length : $\mathbf{N}^* \rightarrow \mathbf{N}$

1. Is  $A$  generated?
2. Determine a minimal set of generators.
3. Does this minimal set freely generate  $A$ ?

# Summary

- ▶ Homomorphisms, epi-, mono-, isomorphisms.
- ▶ Homomorphic and polymorphic Abstract Data Types.
- ▶ Congruence, quotient algebra.
- ▶ Term generated and freely generated algebras.

## Exercises

1. Consider the  $\Sigma$ -algebra  $A$  of natural numbers with 0 and addition. Show that a function  $\varphi: \mathbf{N} \rightarrow \mathbf{N}$  is an endomorphism on  $A$  if and only if there is a number  $k \in \mathbf{N}$  such that  $\varphi(n) = k * n$  for all  $n \in \mathbf{N}$ . How are the automorphisms on  $A$  characterised?
2. Extend the signature of 1. by a unary successor operation and let  $B$  be the extension of  $A$  by the usual successor function on  $\mathbf{N}$ . Show that the only homomorphism on  $B$  is the identity.
3. Let  $\Sigma$  be the signature of Example 1. Consider the  $\Sigma$ -algebra  $C$  of finite lists of natural numbers where 0 is interpreted by the empty list and the binary operation is interpreted by concatenation of lists. Is the operation of reversing a list a homomorphism on  $C$ ?
4. Define an epimorphism from  $C$  to  $A$  and a monomorphism from  $A$  to  $C$ .
5. Which of the algebras  $A$ ,  $B$  and  $C$  are generated respectively freely generated?

## Exercises (ctd.)

6. Prove that homomorphisms are closed under composition.
7. (Part of the proof of the Homomorphism Theorem) Let  $A, B$  algebras for a signature  $\Sigma$  and let  $\varphi: A \rightarrow B$  be a homomorphism. For simplicity, let us assume that  $\Sigma$  has only one sort.

Define a binary relation  $\sim_\varphi$  on  $A$  by

$$a \sim_\varphi b \quad :\Leftrightarrow \quad \varphi(a) = \varphi(b).$$

Show that  $\sim_\varphi$  is a congruence on  $A$ .

# Specifications

- ▶ Loose specifications
- ▶ The Loewenheim-Skolem Theorem
- ▶ Initial specifications
- ▶ Modularisation: Combining specifications
- ▶ Specification languages
- ▶ Summary and Exercises

## Loose specifications and their models

Given

- $\Sigma$  a signature
- $\Phi$  a set of closed  $\Sigma$ -formulas, called **axioms**.

The pair  $(\Sigma, \Phi)$  is called a *loose specification*.

$$\text{Mod}_{\Sigma}(\Phi) \quad :\equiv \quad \{A \in \text{Alg}(\Sigma) \mid A \models \Phi\}$$

is the *class of models of the loose specification*  $(\Sigma, \Phi)$ .

A loose specification is *consistent* if it has a model, that is,  $\text{Mod}_{\Sigma}(\Phi)$  is nonempty.

## A consistent loose specification defines an ADT

### Theorem

If  $(\Sigma, \Phi)$  is a consistent loose specification, then the class of its models,  $\text{Mod}_\Sigma(\Phi)$ , is an abstract data type.

**Proof.** Since  $(\Sigma, \Phi)$  is consistent,  $\text{Mod}_\Sigma(\Phi)$  is nonempty.

Assume  $A \in \text{Mod}_\Sigma(\Phi)$  and  $A \simeq B$ .

We have to show  $B \in \text{Mod}_\Sigma(\Phi)$ .

Since  $A \in \text{Mod}_\Sigma(\Phi)$  we have  $A \models \Phi$ , i.e.  $A \models P$  for all  $P \in \Phi$ .

Since isomorphic algebras satisfy the same formulas (see the next but one slide) it follows that  $B \models P$  for all  $P \in \Phi$  as well.

Hence  $B \models \Phi$ , i.e.  $B \in \text{Mod}_\Sigma(\Phi)$ .

## Homomorphisms preserve the value of a term

### Theorem

Let  $\varphi: A \rightarrow B$  be a homomorphism between  $\Sigma$ -algebras and  $\alpha: X \rightarrow A$  a variable assignment. Then for every term  $t \in \mathbb{T}(\Sigma, X)$  we have

$$\varphi(t^{A, \alpha}) = t^{B, \varphi \circ \alpha}$$

In particular, if  $P$  is closed, then

$$\varphi(t^A) = t^B$$

**Proof.** Structural induction on  $t$ .

## Isomorphic algebras satisfy the same formulas

**Theorem**

Let  $\varphi: A \rightarrow B$  be an isomorphism between  $\Sigma$ -algebras. Then for every formula  $P \in \mathcal{L}(\Sigma, X)$  and every assignment  $\alpha: X \rightarrow A$  we have

$$A, \alpha \models P \quad \text{iff} \quad B, \varphi \circ \alpha \models P$$

In particular, if  $P$  is closed, then

$$A \models P \quad \text{iff} \quad B \models P$$

**Proof.** Structural induction on the formula  $P$ .

## Example: Specifying the Booleans

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	boole
<b>Constants</b>	T, F: boole
<b>Operations</b>	not: boole $\rightarrow$ boole and, or: boole $\times$ boole $\rightarrow$ boole

We wish to give a loose specification of the algebra of Boolean values T and F.

## First attempt

$$\Phi = \{P_1, \dots, P_6\}$$

$$P_1 \quad := \quad \text{not}(\text{T}) = \text{F}$$

$$P_2 \quad := \quad \text{not}(\text{F}) = \text{T}$$

$$P_3 \quad := \quad \text{and}(\text{T}, \text{T}) = \text{T}$$

$$P_4 \quad := \quad \forall x (\text{and}(\text{F}, x) = \text{F})$$

$$P_5 \quad := \quad \forall x (\text{and}(x, \text{F}) = \text{F})$$

$$P_6 \quad := \quad \forall x, y (\text{or}(x, y) = \text{not}(\text{and}(\text{not}(x), \text{not}(y))))$$

The intended model of  $(\Sigma, \Phi)$ 

<b>Algebra</b>	Boole
<b>Carriers</b>	$\mathbf{B} := \{\mathbf{T}, \mathbf{F}\}$
<b>Constants</b>	$\mathbf{T}, \mathbf{F}$
<b>Operations</b>	$\neg: \mathbf{B} \rightarrow \mathbf{B}$ ( <i>negation</i> ) $\wedge: \mathbf{B} \times \mathbf{B} \rightarrow \mathbf{B}$ ( <i>conjunction</i> ) $\vee: \mathbf{B} \times \mathbf{B} \rightarrow \mathbf{B}$ ( <i>disjunction</i> )

Another model of  $(\Sigma, \Phi)$ 

<b>Algebra</b>	$\text{Pow}(\mathbf{N})$
<b>Carriers</b>	$\mathcal{P}(\mathbf{N}) := \{A \mid A \subseteq \mathbf{N}\}$ , the powerset of $\mathbf{N}$
<b>Constants</b>	$\mathbf{N}, \emptyset$
<b>Operations</b>	$\mathbf{N} \setminus \cdot : \mathcal{P}(\mathbf{N}) \rightarrow \mathcal{P}(\mathbf{N})$ (complement) $\cap : \mathcal{P}(\mathbf{N}) \times \mathcal{P}(\mathbf{N}) \rightarrow \mathcal{P}(\mathbf{N})$ (intersection) $\cup : \mathcal{P}(\mathbf{N}) \times \mathcal{P}(\mathbf{N}) \rightarrow \mathcal{P}(\mathbf{N})$ (union)

The ADT  $\text{Mod}_{\Sigma}(\Phi)$  is polymorphic since it contains the non-isomorphic algebras  $\text{Boole}$  and  $\text{Pow}(\mathbf{N})$ .

Why are they non-isomorphic?

## Killing unwanted models

Add further axioms so that the  $\Sigma$ -algebra `Boole` becomes the 'only' model up to isomorphism:

"every element is either true or false" (thus 'killing' the model `Pow(N)`)

"T and F are different" (thus killing the one-element algebra).

$P_7 \quad :\equiv$

$P_8 \quad :\equiv$

$\text{Mod}_\Sigma(\Phi \cup \{P_7, P_8\})$  is a monomorphic ADT containing (up to isomorphism) only the algebra `Boole`.

## Example: Specifying the natural numbers

<b>Signature</b>	$\Sigma$
<b>Sorts</b>	nat
<b>Constants</b>	0: nat
<b>Operations</b>	succ: nat $\rightarrow$ nat +: nat $\times$ nat $\rightarrow$ nat

$\mathbf{N}_{0S+}$  :=  $\Sigma$ -algebra of natural numbers with constant 0,  
 successor function succ:  $\mathbf{N} \rightarrow \mathbf{N}$ , succ( $n$ ) :=  $n + 1$ ,  
 addition +:  $\mathbf{N} \times \mathbf{N} \rightarrow \mathbf{N}$ .

Try to characterise the  $\Sigma$ -algebra  $\mathbf{N}_{0S+}$  up to isomorphism by a loose specification  $(\Sigma, \Phi)$  with a suitable set of axioms  $\Phi$ .

## Specifying the natural numbers (ctd.)

Start with putting into  $\Phi$  formulas  $P_1, \dots, P_5$  expressing

1. 0 is not a successor,
2. succ is injective (one-to-one)
3. every number is either 0 or a successor,
- 4/5. addition can be defined from 0 and succ by primitive recursion in the usual way.

## An unwanted model

'Unwanted' model of  $A$  of  $(\Sigma, P_1, \dots, P_5)$ :

$$A_{\text{nat}} := \{0\} \times \mathbf{N} \cup \{1\} \times \mathbf{Z} =$$

$$\{(0, 0), (0, 1), (0, 2), \dots\}$$

$$\cup \{\dots, (1, -2), (1, -1), (1, 0), (1, 1), (1, 2), \dots\}$$

$$0^A := (0, 0),$$

$$\text{succ}^A((i, n)) := (i, n + 1),$$

$$+^A((i, n), (j, m)) := (\max(i, j), n + m)$$

Killing the unwanted model:

$$P_6 \quad \equiv \quad \forall x (x + x = x \rightarrow x = 0)$$

Did we succeed in specifying the intended model  $\mathbf{N}_{0S+}$ ?

## The Loewenheim-Skolem Theorem

We will never succeed in characterising  $\mathbb{N}_{0S+}$  up to isomorphism by more and more axioms, due to the following theorem:

### **Theorem (Loewenheim-Skolem)**

If a loose specification  $(\Sigma, \Phi)$  has a countably infinite model  $A$ , then it also has an uncountable model  $B$ . In particular  $A$  and  $B$  are non-isomorphic, and therefore the abstract data type  $\text{Mod}_{\Sigma}(\Phi)$  is polymorphic.

## Displaying a loose specification

**Loose Spec****Sorts**         $\text{nat}$ **Constants**    $0: \text{nat}$ **Operations**    $\text{succ}: \text{nat} \rightarrow \text{nat}$   
 $+: \text{nat} \times \text{nat} \rightarrow \text{nat}$ **Variables**     $x, y: \text{nat}$ **Axioms**         $0 \neq \text{succ}(x)$   
 $\text{succ}(x) = \text{succ}(y) \rightarrow x = y$   
 $x = 0 \vee \exists y (x = \text{succ}(y))$   
 $x + 0 = x$   
 $x + \text{succ}(y) = \text{succ}(x + y)$

# Overcoming the weakness of loose specifications

In order to increase the expressive power of specifications we restrict their models to algebras that are **initial** in the class of all models of the loose specification.

The property of being initial (which will be explained next) pins down an algebra up to isomorphism.

In order to ensure that initial models exist we will restrict the axioms to equations.

## Initial specifications – Definition

An *Initial Specification*

$$\text{Init-Spec}(\Sigma, E)$$

consists of a signature  $\Sigma$  and a set of equations  $E$  between  $\Sigma$ -terms.

A  $\Sigma$ -algebra  $A$  is a **model** of  $\text{Init-Spec}(\Sigma, E)$  if  $A$  is **initial** in  $\text{Mod}_{\Sigma}(\forall E)$ . This means:

- (i) (Model property)  $A \in \text{Mod}_{\Sigma}(\forall E)$ , that is,  $A \models \forall \vec{x} (t_1 = t_2)$  for all equation  $t_1 = t_2 \in E$  where  $\vec{x} = \text{FV}(t_1 = t_2)$ .
- (ii) (Initiality) For any algebra  $B \in \text{Mod}_{\Sigma}(\forall E)$  there exists exactly one homomorphism  $\varphi : A \rightarrow B$ .

## Initial specifications – Notation

$\text{Init-Mod}_\Sigma(E)$  := class of models of the initial specification  
 $\text{Init-Spec}(\Sigma, E)$ .

If  $A \in \text{Init-Mod}_\Sigma(E)$  one also says

“ $A$  is an initial model of  $E$ ”.

If  $A$  is a  $\Sigma$ -algebra, then the task to

“give an initial specification of  $A$ ”

means to find an initial specification  $\text{Init-Spec}(\Sigma, E)$  such that  
 $A \in \text{Init-Mod}_\Sigma(E)$ .

## Initial Model Theorem

1. Every initial specification  $\text{Init-Spec}(\Sigma, E)$  has a model.
2. The class  $\text{Init-Mod}_{\Sigma}(E)$  of all initial models is a monomorphic ADT.

## Term algebra

Let  $\Sigma$  be a signature.

The **closed term algebra**  $T(\Sigma)$  has as carrier the set of all closed  $\Sigma$ -terms.

Constants and operations are interpreted syntactically, that is:

- ▶  $c^{T(\Sigma)} := c$  for every constant  $c$ .
- ▶  $f^{T(\Sigma)}(t_1, \dots, t_n) := f(t_1, \dots, t_n)$ .

We will see that the term algebra is useful for constructing initial models.

## Construction of an initial model

We prove part 1 of the Initial Model Theorem.

Let  $\text{Init-Spec}(\Sigma, E)$  be an initial specification.

We define a relation  $\sim_E$  on closed  $\Sigma$ -terms by

$$t_1 \sim_E t_2 \quad :\Leftrightarrow \quad \forall E \models t_1 = t_2$$

### Theorem

$\sim_E$  is a congruence on the closed term algebra  $\mathbb{T}(\Sigma)$ .

The quotient algebra

$$\mathbb{T}_E(\Sigma) := \mathbb{T}(\Sigma) / \sim_E$$

is a model of  $\text{Init-Spec}(\Sigma, E)$ .

For every  $A \in \text{Mod}_\Sigma(\forall E)$  the unique homomorphism  $\varphi: \mathbb{T}_E(\Sigma) \rightarrow A$  is given by  $\varphi_s([t]_E) = t^A$ .

## Init-Mod $_{\Sigma}(E)$ is a monomorphic ADT

Now we prove part 2 of the Initial Model Theorem.

We have to show

- (a) Init-Mod $_{\Sigma}(E)$  is an ADT.
- (b) Init-Mod $_{\Sigma}(E)$  is monomorphic.

We only prove (a).

The proof of (b) is similar and left as an exercise.

## Proof that $\text{Init-Mod}_\Sigma(E)$ is an ADT

Assume  $A \in \text{Init-Mod}_\Sigma(E)$  and  $A \simeq B$ . We have to show  $B \in \text{Init-Mod}_\Sigma(E)$ . Let  $\varphi: B \rightarrow A$  be an isomorphism.

Since  $\text{Mod}_\Sigma(\forall E)$  is an ADT and  $A \in \text{Mod}_\Sigma(\forall E)$  it follows  $B \in \text{Mod}_\Sigma(\forall E)$ .

In order to show that  $B$  is initial in  $\text{Mod}_\Sigma(\forall E)$  we take an arbitrary  $\Sigma$ -algebra  $C \in \text{Mod}_\Sigma(\forall E)$  and show that there is exactly one homomorphism from  $B$  to  $C$ .

Since  $A$  is initial for in  $\text{Mod}_\Sigma(\forall E)$ , there is a unique homomorphism  $\chi: A \rightarrow C$ . Then  $\chi \circ \varphi: B \rightarrow C$  is a homomorphism.

We prove that any other homomorphism from  $B$  to  $C$  coincides with  $\chi \circ \varphi$ . So, let  $\psi: B \rightarrow C$  a homomorphism. Since  $\psi \circ \varphi^{-1}: A \rightarrow C$  is a homomorphism we may use the initiality of  $A$  to conclude that  $\psi \circ \varphi^{-1} = \chi$ . Hence  $\psi = \chi \circ \varphi$ .

## Characterisation Theorem

Let  $\mathbb{T}_E(\Sigma)$  be an initial specification. Then for any  $\Sigma$ -algebra  $A$  the following conditions are equivalent:

- (i)  $A$  is a model of  $\text{Init-Spec}(\Sigma, E)$ .
- (ii)  $A$  is initial in  $\text{Mod}_\Sigma(\forall E)$  (i.e.  $A$  is an initial model of the loose specification  $(\Sigma, \forall E)$ ).
- (iii)  $A \simeq \mathbb{T}_E(\Sigma)$ .
- (iv)  $A$  is generated and for any two closed  $\Sigma$ -terms  $t_1, t_2$  of the same sort we have
 
$$A \models t_1 = t_2 \quad \Leftrightarrow \quad \forall E \models t_1 = t_2$$
 (i.e.  $t_1$  and  $t_2$  have the same value in  $A$  iff they have the same value in all models of  $\forall E$ ).
- (v)  $A$  is a generated model of  $\forall E$  and for any two closed  $\Sigma$ -terms  $t_1, t_2$  of the same sort we have

$$A \models t_1 = t_2 \quad \Rightarrow \quad \forall E \models t_1 = t_2$$

## Example

<b>Init Spec</b>	NAT
<b>Sorts</b>	nat
<b>Constants</b>	0: nat
<b>Operations</b>	succ: nat $\rightarrow$ nat +: nat $\times$ nat $\rightarrow$ nat
<b>Variables</b>	x, y: nat
<b>Equations</b>	$x + 0 = x$ $x + \text{succ}(y) = \text{succ}(x + y)$

The elements of the carrier set of  $\mathbb{T}_E(\Sigma)$  are the equivalence classes  $[0]$ ,  $[\text{succ}(0)]$ ,  $[\text{succ}(\text{succ}(0))]$ ,  $\dots$

One has for instance

$$\begin{aligned} [0] &= \{0, 0 + 0, (0 + 0) + 0, \dots\} \\ &= \text{the set of closed terms built from } 0 \text{ and } + \end{aligned}$$

$$\begin{aligned} [\text{succ}(0)] &= \{\text{succ}(0), \text{succ}(0) + 0, 0 + \text{succ}(0), \dots\} \\ &= \text{the set of closed terms built from } 0 \text{ and } + \\ &\quad \text{and exactly one occurrence of succ} \end{aligned}$$

Furthermore, for any  $n, m \in \mathbf{N}$  we have

$[\text{succ}^n(0) + \text{succ}^m(0)] = [\text{succ}^{n+m}(0)]$ , and from this it easily follows that for any two closed terms  $t_1, t_2$  we have  $\mathbf{N} \models t_1 = t_2$  iff the equation  $t_1 = t_2$  follows from the equations  $x = 0$  and  $x + \text{succ}(y) = \text{succ}(x + y)$ .

Therefore, NAT is an initial specification of natural numbers with zero and addition.

## Initial specification of finite sets

<b>Init Spec</b>	<i>SET</i>
<b>Sorts</b>	nat, set
<b>Constants</b>	0: nat emptyset: set
<b>Operations</b>	succ: nat $\rightarrow$ nat insert: set $\times$ nat $\rightarrow$ set
<b>Variables</b>	$x, y$ : nat, $s$ : set
<b>Equations</b>	insert(insert( $s, x$ ), $x$ ) = insert( $s, x$ ) insert(insert( $s, x$ ), $y$ ) = insert(insert( $s, y$ ), $x$ )

Let  $A$  be the classical algebra of finite set of natural numbers with the obvious interpretation of the constants and operation.

## Proving is that $A$ is a model of $SET$

We use the Characterisation Theorem, part (iv), to show that  $A$  is a model of the initial specification  $SET$ , or, in other words,  $SET$  is an initial specification of the algebra of finite sets: One easily sees that

- ▶  $A$  is a generated model of  $SET$ .
- ▶ If two closed terms have the same value in  $A$ , then they have the same value in all models of the equations.

## Exercise

Suppose we add an operation  $\text{select}: \text{set} \rightarrow \text{nat}$  and the equations

$$\text{select}(\text{emptyset}) = 0$$

$$\text{select}(\text{insert}(s, x)) = x$$

What is the problem with the extended specification?

### Example: Text editor

- ▶ `write(x)`: insert the character `x` immediately to the left of the cursor;
- ▶ `▷`: move the cursor one position to the right;
- ▶ `◁`: move the cursor one position to the left;
- ▶ `del`: delete the character immediately to the right of the cursor.

## The editor in action

Command	Display	Internal representation
	<i>edir or</i>	([r, i, d, e], [o, r])
◁	<i>edi ror</i>	([i, d, e], [r, o, r])
del	<i>edi or</i>	([i, d, e], [o, r])
write t	<i>edit or</i>	([t, i, d, e], [o, r])

## Initial specification of the editor

<b>Init Spec</b>	<i>EDITOR</i>
<b>Sorts</b>	char, charlist, file
<b>Constants</b>	newfile: file $a, \dots, z, \_:$ char nil: charlist
<b>Operations</b>	cons: char $\times$ charlist $\rightarrow$ charlist cf: charlist $\times$ charlist $\rightarrow$ file write: char $\times$ file $\rightarrow$ file $\triangleleft$ : file $\rightarrow$ file $\triangleright$ : file $\rightarrow$ file del: file $\rightarrow$ file

## Initial specification of the editor (ctd.)

**Variables**  $x$ : char,  $l, r$ : charlist

**Equations**  $\text{newfile} = \text{cf}(\text{nil}, \text{nil})$

$\text{write}(x, \text{cf}(l, r)) = \text{cf}(\text{cons}(x, l), r)$

$\triangleleft (\text{cf}(\text{nil}, r)) = \text{cf}(\text{nil}, r)$

$\triangleleft (\text{cf}(\text{cons}(x, l), r)) = \text{cf}(l, \text{cons}(x, r))$

$\triangleright (\text{cf}(l, \text{nil})) = \text{cf}(l, \text{nil})$

$\triangleright (\text{cf}(l, \text{cons}(x, r))) = \text{cf}(\text{cons}(x, l), r)$

$\text{del}(\text{cf}(l, \text{nil})) = \text{cf}(l, \text{nil})$

$\text{del}(\text{cf}(l, \text{cons}(x, r))) = \text{cf}(l, r)$

## Combining specifications

Union       $\text{Spec}_1 + \text{Spec}_2$

$A$  is a model of  $\text{Spec}_1 + \text{Spec}_2$  iff  $A|_{\Sigma_1}$  is a model of  $\text{Spec}_1$  and  $A|_{\Sigma_2}$  is a model of  $\text{Spec}_2$ .

Restriction       $\text{Spec}|_{\Sigma_0}$

$A$  is a model of  $\text{Spec}|_{\Sigma_0}$  iff  $A = B|_{\Sigma_0}$  for some model  $B$  of  $\text{Spec}$ .

# Other fundamental construction principles

- ▶ **Renaming**
- ▶ **Parametrization**
- ▶ **Modularization**
- ▶ **Inheritance**

## Example: Union

<b>Loose Spec</b>	MINUS
<b>Sorts</b>	nat
<b>Constants</b>	0: nat
<b>Operations</b>	succ: nat $\rightarrow$ nat - : nat $\times$ nat $\rightarrow$ nat
<b>Variables</b>	x, y: nat
<b>Equations</b>	$x - 0 = x$ $\text{succ}(x) - \text{succ}(y) = x - y$

The models of the specification NAT + MINUS are the  $\{\text{nat}, 0, \text{succ}, +, -\}$ -algebras, where the natural numbers, addition and  $n - m$  for  $n \geq m$  have their standard meaning, but the result of  $n - m$  for  $n < m$  can be any natural number.

## Example: Restriction

In the initial specification EDITOR the subsignature  $\Sigma_0$  of interest to the user consists of the sorts char and file, the constants  $a, \dots, z, \_:$  char and newfile, as well as the operations

$\triangleright, \triangleleft, \text{del}: \text{file} \rightarrow \text{file}$

$\text{write}: \text{char} \times \text{file} \rightarrow \text{file}.$

The other sorts, constants and operations are auxiliaries needed for constructing files and writing the specification. Therefore the restricted specification

$$\text{EDITOR}_{|\Sigma_0}$$

describes the algebra that is handed over to the user.

## Specification Languages

**VDM, Z** Use set-theoretic notation. Widely used in industry.

**ASL** Kernel language for algebraic specifications.

**Extended ML** Specification language for functional programming languages, in particular ML.

**Spectrum** Very general, based on partial algebras, higher order constructs and polymorphism.

**Larch** State oriented, contains an elaborate proof checker.

# Specification languages (ctd.)

CCS, CSP Formal languages for specifying concurrent processes.

UML Design and modeling language for object oriented programming.

CASL, CSP CASL Common Algebraic Specification Language.  
Machine support by the interactive Theorem Prover Isabelle/HOL. Integration of Process Algebra.

## Summary

- ▶ Isomorphic algebras have the same properties, i.e. they satisfy the same closed formulas.
- ▶ *Loose specifications*: Arbitrary axioms allowed. Every algebra satisfying the axioms is a model. The class of all models of a consistent loose specification forms an ADT. By the Löwenheim-Skolem Theorem, loose specifications usually cannot pin down ADTs up to isomorphism, that is, the model class is a *polymorphic* ADT.

## Summary (ctd.)

- ▶ Initial algebras, uniqueness up to isomorphism of initial algebras.
- ▶ *Initial Specifications*: Only *equations* allowed. Every algebra which is initial in the class of all loose models of a specification is a model of an initial specification. The class of all models of an initial specification forms an ADT. Initial specifications do pin down ADTs up to isomorphism, that is, the model class is a *monomorphic* ADT. A model of an initial specification can be constructed as a quotient of the term algebra.
- ▶ Generators and observers,
- ▶ *Modularisation*: Structuring specifications using, for example, restriction (hiding) and union of specifications,
- ▶ Specification Languages.

## Exercises

- ▶ Determine for the specifications BOOLE, SET, EDITOR generators and observers. Are the corresponding algebras freely generated by the generators?
- ▶ Let  $A$  be the algebra of natural numbers with 0 and addition and  $C$  the algebra of lists of natural numbers with the empty list and concatenation of lists (so  $A$  and  $C$  are algebras over the same signature). Show that there is a bijection between  $A$  and  $C$ , but no isomorphism.

*Hint:* For showing that  $A$  and  $C$  are not isomorphic use the Theorem that isomorphic algebras satisfy the same formulas.

## Exercise

- ▶ Give an initial specification of FIFO (first-in-first-out) queues of natural numbers. The signature should contain (among other things) the operations
  - $\text{snoc} : \text{queue} \rightarrow \text{nat} \rightarrow \text{queue}$ , inserting an element at the end of a queue,
  - $\text{head} : \text{queue} \rightarrow \text{nat}$ , computing the first element of a nonempty queue,
  - $\text{tail} : \text{queue} \rightarrow \text{queue}$ , computing the tail of a nonempty queue (first element removed),
  - $\text{member} : \text{nat} \rightarrow \text{queue} \rightarrow \text{boole}$ , testing membership,
  - $\text{length} : \text{queue} \rightarrow \text{nat}$ , computing the length of a queue,
  - $\text{isempty} : \text{queue} \rightarrow \text{boole}$ , testing whether a queue is empty.

Determine constructors and observers. Are the constructors free?

## Exercise

- ▶ Let  $\Sigma$  be the signature with one sort, two constants, 0 and 1, and two binary operations,  $+$  and  $*$ . Let  $R$  be the  $\Sigma$ -algebra of real numbers with 0, 1, addition and multiplication.
  - (a) Show that “ $x < y$ ” is expressible by a  $\Sigma$ -formula.
  - (b) Let  $\varphi: R \rightarrow R$  be an automorphism. Show that  $\varphi(q) = q$  for all rational numbers  $q$ .
  - (c) Show that the only automorphism on  $R$  is the identity.

*Hint for (c).* Use parts (a) and (b) to show that for all rationals  $q$  and all reals  $r$  we have  $q < \varphi(r)$  if and only if  $q < r$ . This clearly implies  $\varphi(r) = r$  for all reals  $r$ .

*Remark:* On the  $\Sigma$ -algebra  $C$  of complex numbers there exist exactly two automorphisms: The identity, and the mapping sending a complex number  $z = x + iy$  to its conjugate complex  $\bar{z} = x - iy$ .

# Term rewriting

- ▶ Equational logic
- ▶ Term rewriting systems
- ▶ Termination
- ▶ Confluence
- ▶ Summary and Exercises

# Term rewriting = automatizing equational logic

- ▶ We study a specific proof calculus for reasoning with equations.
- ▶ Term rewriting is an automatic proof procedure for equational reasoning.
- ▶ Automatic equational reasoning is important in
  - theorem proving: most proving problems are to a large extent concerned with equations;
  - implementing ADTs given by an initial specification (rapid prototyping).

## Substitutions

Substitutions play an important role in equational reasoning.

A **substitution** is a mapping from variables to terms.

If  $t$  is a term, then  $t\theta$  denotes the term obtained by replacing every variable  $x$  in  $t$  by  $\theta(x)$ .

**Example:**  $\theta := \{0/x, \text{succ}(x)/y\}$ . This means  $\theta(x) = 0$ ,  $\theta(y) = \text{succ}(x)$  and  $\theta(z) = z$  for all other variables  $z$ .

Then  $(x + \text{succ}(x + y))\theta = 0 + \text{succ}(0 + \text{succ}(x))$ .

## The rules of equational logic

**Reflexivity** 
$$\frac{}{t = t}$$

**Symmetry** 
$$\frac{t_1 = t_2}{t_2 = t_1}$$

**Transitivity** 
$$\frac{t_1 = t_2 \quad t_2 = t_3}{t_1 = t_3}$$

**Compatibility** 
$$\frac{t_1 = t'_1 \quad \dots \quad t_n = t'_n}{f(t_1, \dots, t_n) = f(t'_1, \dots, t'_n)}$$

**Instance** 
$$\frac{t_1 = t_2}{t_1\theta = t_2\theta}$$

$$\vdash_E t_1 = t_2$$

means  $t_1 = t_2$  is derivable by the rules above starting from equations in  $E$ .

## Example

$$E := \{ x + 0 = x, x + \text{succ}(y) = \text{succ}(x + y) \}$$

$$\vdash_E 0 + \text{succ}(0) = \text{succ}(0)$$

$$\frac{\frac{x + \text{succ}(y) = \text{succ}(x + y)}{0 + \text{succ}(0) = \text{succ}(0 + 0)} \text{Inst} \quad \frac{\frac{x + 0 = x}{0 + 0 = 0} \text{Inst}}{\text{succ}(0 + 0) = \text{succ}(0)} \text{Comp}}{0 + \text{succ}(0) = \text{succ}(0)} \text{Trans}$$

## Soundness Theorem

If  $\vdash_E t_1 = t_2$  then  $\forall E \vdash_m \forall(t_1 = t_2)$  and therefore  $\forall E \models \forall(t_1 = t_2)$ .

### Proof.

Clearly, the rules of equational logic can be mimicked by rules of minimal logic (with equality).

Hence  $\forall E \models \forall(t_1 = t_2)$ , by the soundness theorem for natural deduction.

Of course, it is also easy to see directly that  $\vdash_E t_1 = t_2$  implies  $\forall E \models \forall(t_1 = t_2)$ .

## Completeness Theorem (Birkhoff)

If  $\forall E \models \forall(t_1 = t_2)$  then  $\vdash_E t_1 = t_2$ .

**Proof (sketch).**  $\Sigma :=$  the signature of  $E \cup \{t_1 = t_2\}$

$V :=$  the set of all variables for the sorts of  $\Sigma$ .

Then  $\mathbb{T}(\Sigma, V)$  is the set of all  $\Sigma$ -terms.

$$t_1 =_E t_2 \quad :\Leftrightarrow \quad \vdash_E t_1 = t_2$$

$=_E$  is a congruence on the term algebra  $\mathbb{T}(\Sigma, V)$ .

Set  $A := \mathbb{T}(\Sigma, V) / =_E$ . One can show

$$A \models \forall(t_1 = t_2) \quad \Leftrightarrow \quad \vdash_E t_1 = t_2 \quad (++)$$

From  $(++)$ , ' $\Leftarrow$ ' it follows that  $A$  is a model of  $\forall E$ .

Now:

Assume  $\forall E \models \forall(t_1 = t_2)$ . Then  $A \models \forall(t_1 = t_2)$ , since  $A$  is a model of  $\forall E$ . Consequently  $\vdash_E t_1 = t_2$ , by  $(++)$ ,  $\Rightarrow$ .

## Term rewriting systems

A **term rewriting system** over a signature  $\Sigma$  is a finite set  $R$  of **rewrite rules**  $l \mapsto r$ , where  $r$  and  $l$  are  $\Sigma$ -terms, such that

- (i)  $l$  is not a variable,
- (ii)  $FV(r) \subseteq FV(l)$ .

$$t \rightarrow_R t' \quad :\Leftrightarrow \quad t \equiv u\{l\theta/x\} \text{ and } t' \equiv u\{r\theta/x\}$$

for some rewrite rule  $l \mapsto r \in R$ ,  
 some  $\Sigma$ -term  $u$   
 with exactly one occurrence  
 of some variable  $x$ , and  
 some substitution  $\theta$

$\rightarrow_R$  is called the **term rewriting relation generated by  $R$** .

## Reduction sequences

$$t \rightarrow_R^* t' \quad :\Leftrightarrow \quad t \rightarrow_R \dots \rightarrow_R t'$$

$$t \leftrightarrow_R t' \quad :\Leftrightarrow \quad t \rightarrow_R t' \text{ or } t' \rightarrow_R t$$

$$t \simeq_R t' \quad :\Leftrightarrow \quad t \leftrightarrow_R \dots \leftrightarrow_R t'$$

Any finite or infinite sequence  $t_0 \rightarrow_R t_1 \rightarrow_R \dots$  is called a **reduction sequence**.

## Normal forms

$t$  is in **normal form** w.r.t.  $R$  if it cannot be rewritten, i.e.  $t \not\rightarrow_R t'$  for any  $t'$ .

$t'$  is a **normal form of**  $t$ , or  $t$  **normalizes to**  $t'$  if  $t \rightarrow_R^* t'$  and  $t'$  is in normal form.

# The rewrite system defined by a set of equations

$R := \{l \mapsto r \mid l = r \in E\}$  is called the **term rewriting system defined by  $E$** .

We write  $t \rightarrow_E t'$  instead of  $t \rightarrow_R t'$ .

## Example

$$E := \{ x + 0 = x, x + \text{succ}(y) = \text{succ}(x + y) \}.$$

$$\begin{aligned} 0 + (0 + \text{succ}(0)) &\rightarrow_E 0 + \text{succ}(0 + 0) \\ &\rightarrow_E 0 + \text{succ}(0) \\ &\rightarrow_E \text{succ}(0 + 0) \\ &\rightarrow_E \text{succ}(0) \end{aligned}$$

Exercise: normalize  $\text{succ}(0 + 0) + 0$

Normal forms:  $\text{succ}^n(0)$

(that is  $0, \text{succ}(0), \text{succ}(\text{succ}(0)) \dots$ )

$$0 + (0 + \text{succ}(0)) \simeq_E \text{succ}(0 + 0) + 0.$$

Exercise: normalize  $0 + (0 + \text{succ}(0))$  using a different reduction

# The equivalence of equational logic and term rewriting

## Theorem

For a set  $E$  of equations and closed terms  $t, t'$  the following assertions are equivalent:

- (i)  $\forall E \models t = t'$
- (ii)  $\forall E \vdash_c t = t'$
- (iii)  $\forall E \vdash_m t = t'$
- (iv)  $\vdash_E t = t'$
- (v)  $t \simeq_E t'$

# Termination

A term rewriting system  $R$  is **terminating** if there is no infinite reduction sequence

$$t_0 \rightarrow_R t_1 \rightarrow_R \dots$$

Terminating term rewriting systems are often called **strongly normalising** or **noetherian** (after Emmy Noether).

## Emmy Noether



E Noether (1882 - 1935)

## Weak normalisation

In a terminating term rewriting system every term has a normal form, but the converse is not true. For example

$$R := \left\{ \begin{array}{l} x + 0 \mapsto x, \\ x + \text{succ}(y) \mapsto \text{succ}(x + y), \\ 0 + y \mapsto y + 0 \end{array} \right\}$$

$R$  is not terminating, but every term has a normal form.

By removing the last rule the term rewriting system becomes terminating.

Term rewriting systems where every term has a normal form are often called **weakly normalising**.

## Proving termination

A function  $\mu$  mapping  $\Sigma$ -terms to natural numbers such that

$$t \rightarrow_R t' \quad \Rightarrow \quad \mu(t) > \mu(t')$$

is called a **termination measure** for the term rewriting system  $R$ .

**Lemma.** Every rewrite system that has a termination measure is terminating.

## A simple termination proof method

**Lemma.** Let  $R$  be a term rewriting system such for every rule  $l \mapsto r$  in  $R$

$r$  is shorter than  $l$ ,

every variable  $x \in \text{FV}(r)$  occurs in  $r$  at most as often as it occurs in  $l$ .

Then  $R$  is terminating.

**Proof.** The assumptions imply that the length of terms is a termination measure.

## Exercises

Which of the following term rewriting systems are terminating?

$$R_1 := \{ x - 0 \mapsto x, \text{succ}(x) - \text{succ}(y) \mapsto x - y \}$$

$$R_2 := \{ f(g(x), y) \mapsto f(y, y) \}$$

$$R_3 := \{ x + 0 \mapsto x, x + \text{succ}(y) \mapsto \text{succ}(x + y) \}$$

## Termination by strictly monotonic algebras

**Theorem**

Let  $R$  be a term rewriting system over  $\Sigma$ .

Let  $A$  be a  $\Sigma$ -algebra such that each carrier set  $A_s$  is a subset of the natural numbers.

Assume that  $f^A$  is strictly monotone for every operation  $f$ , that is,

$$\begin{aligned} n_i > n'_i &\Rightarrow \\ f^A(n_1, \dots, n_i, \dots, n_k) &> f^A(n_1, \dots, n'_i, \dots, n_k) \end{aligned}$$

Assume further that  $l^{A,\alpha} > r^{A,\alpha}$  for every rule  $l \mapsto r \in R$  and every variable assignment  $\alpha$ .

Then  $R$  is terminating.

**Proof.** Show that  $\mu(t) := t^{A,\alpha}$  is a termination measure, where  $\alpha$  is an arbitrary variable assignment.

## Example

$$R := \{ \begin{array}{l} x + 0 \mapsto x, \\ x + \text{succ}(y) \mapsto \text{succ}(x + y) \end{array} \}$$

Define algebra  $A$  by  $A := \mathbf{N}$ ,

$$0^A := 1, \text{succ}^A(n) := n + 1, n +^A m := n + 2 * m.$$

Let  $\alpha$  be an arbitrary variable assignment.  $n := \alpha(x)$ ,  $m := \alpha(y)$

$$(x + 0)^{A,\alpha} = n +^A 0^A = n + 2 * 1 > n = x^{A,\alpha}$$

$$\begin{aligned} (x + \text{succ}(y))^{A,\alpha} &= n +^A \text{succ}^A(m) \\ &= n + 2 * (m + 1) \\ &> n + 2 * m + 1 \\ &= \text{succ}(x + y)^{A,\alpha} \end{aligned}$$

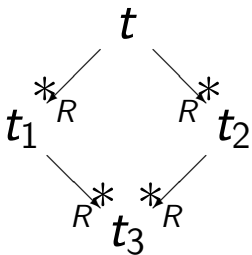
## Confluence

A term rewriting system  $R$  is **confluent** if for all terms  $t, t_1, t_2$ :

if  $t \rightarrow_R^* t_1$  and  $t \rightarrow_R^* t_2$ ,

then there exists  $t_3$  such that

$t_1 \rightarrow_R^* t_3$  and  $t_2 \rightarrow_R^* t_3$ .



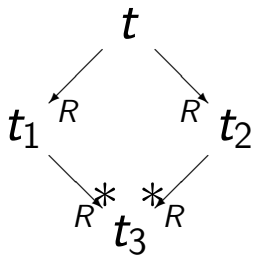
## Local confluence

$R$  is **locally confluent** if for all terms  $t, t_1, t_2$ :

if  $t \rightarrow_R t_1$  and  $t \rightarrow_R t_2$ ,

then there exists  $t_3$  such that

$t_1 \rightarrow_R^* t_3$  and  $t_2 \rightarrow_R^* t_3$ .



## Newman's Lemma

Every terminating and locally confluent term rewriting system is confluent.

**Exercise:** Prove Newman's Lemma.

## Example: confluence vs. local confluence

$$R := \{ a \mapsto b, a \mapsto c, b \mapsto a, b \mapsto d \}$$

$R$  is locally confluent.

$R$  is not confluent, since  $a \rightarrow_R c$  and  $a \rightarrow_R^* d$ , but  $c$  and  $d$  cannot be reduced to a common term.

*Exercises.*

- (a) Add one rewrite rule that makes  $R$  confluent.
- (b) Remove one rewrite rule such that  $R$  becomes confluent.
- (c) Is  $R$  terminating?

# The Church-Rosser property

## Theorem

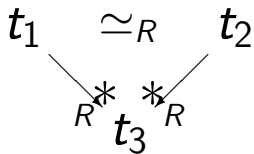
Every confluent term rewriting  $R$  has the

*Church-Rosser property*:

For all  $t_1, t_2$

$$t_1 \simeq_R t_2 \Leftrightarrow$$

there exists  $t_3$  s.t.  $t_1 \rightarrow_R^* t_3$  and  $t_2 \rightarrow_R^* t_3$



## Unique normal forms and decidability

### Lemma

If  $R$  is confluent and terminating term rewriting system then every term  $t$  has a unique normal form  $\text{nf}(t)$ .

### Theorem

Let  $E$  be a system of equations defining a confluent and terminating term rewriting system . Then for all  $t_1, t_2$

$$\forall E \models t_1 = t_2 \quad \Leftrightarrow \quad \text{nf}(t_1) = \text{nf}(t_2)$$

In particular, the relation  $\forall E \models t_1 = t_2$  is decidable.

**Proof.**  $t_1 \simeq_R t_2$  if and only if  $\text{nf}(t_1) = \text{nf}(t_2)$ .

## Rapid prototyping

Let  $\text{Init-Spec}(\Sigma, E)$  be an initial specification such that the term rewriting system associated with it is terminating and confluent.

The algebra  $\text{NF}_E(\Sigma)$  of closed normal terms is defined as follows.

<b>Algebra</b>	$\text{NF}_E(\Sigma)$
<b>Carriers</b>	$\text{NF}_E(\Sigma)_s := \{t \in \text{T}(\Sigma) \mid t \text{ in normal form} \}$
<b>Constants</b>	$c^{\text{NF}_E(\Sigma)} := \text{nf}(c)$
<b>Operations</b>	$f^{\text{NF}_E(\Sigma)}(t_1, \dots, t_n) := \text{nf}(f(t_1, \dots, t_n))$

**Theorem**  $\text{NF}_E(\Sigma)$  is a model of  $\text{Init-Spec}(\Sigma, E)$ . For every closed  $\Sigma$ -term  $t$  we have

$$t^{\text{NF}_E(\Sigma)} = \text{nf}(t)$$

Since a program for computing  $\text{nf}(t)$  can be derived from the rewrite system automatically,  $\text{NF}_E(\Sigma)$  is called a **rapid prototype**.

## Proving confluence: Critical pairs

Let  $l_1 \mapsto r_1$ ,  $l_2 \mapsto r_2$  be variants of rules in  $R$  with  $\text{FV}(l_1) \cap \text{FV}(l_2) = \emptyset$ .

Let  $t$  be a subterm of  $l_1$  which is not a variable, that is  $l_1 \equiv u\{t/x\}$ ,  $x$  fresh occurring in  $u$  exactly once.

Let  $t\theta \equiv l_2\theta$ , where  $\theta$  is a most general unifier.

$$\begin{array}{ccc}
 l_1\theta \equiv (u\theta)\{l_2\theta/x\} & & \\
 \swarrow & & \searrow \\
 r_1\theta & \xleftarrow{R} & (u\theta)\{r_2\theta/x\}
 \end{array}$$

$(r_1\theta, (u\theta)\{r_2\theta/x\})$  is a **critical pair** of  $R$ .

$\text{CP}(R) :=$  the set of all critical pairs of  $R$ .

## Critical Pair Theorem

### Critical Pair Lemma

A term rewriting system  $R$  is locally confluent iff for all critical pairs  $(t_1, t_2)$  of  $R$  there exists a term  $t$  such that  $t_1 \rightarrow_R^* t$  and  $t_2 \rightarrow_R^* t$ .

### Critical Pair Theorem

A terminating term rewriting system  $R$  is confluent iff for all critical pairs  $(t_1, t_2)$  of  $R$  there exists a term  $t$  such that  $t_1 \rightarrow_R^* t$  and  $t_2 \rightarrow_R^* t$ .

In particular, it is decidable whether a terminating term rewriting system is confluent.

**Proof.** Critical Pair Lemma and Newman's Lemma.

**Remark.** For arbitrary term rewriting systems confluence is undecidable (see e.g. Baader/Nipkov).

## Exercise: termination and confluence

$$R := \{ x < x \mapsto F$$

$$0 < \text{succ}(x) \mapsto T$$

$$\text{succ}(x) < 0 \mapsto F$$

$$\text{succ}(x) < \text{succ}(y) \mapsto x < y \}$$

Is  $R$  terminating?

Is  $R$  confluent?

## Knuth-Bendix Completion Algorithm

In order to transform a terminating term rewriting system  $R$  into an equivalent one that is confluent, do the following:

1. Compute  $CP(R)$ . If for all  $(t_1, t_2) \in CP(R)$  there is a  $t$  with  $t_1 \rightarrow_R^* t$  and  $t_2 \rightarrow_R^* t$  then stop (in this case  $R$  is confluent according to the Critical Pair Theorem).
2. For any  $(t_1, t_2) \in CP(R)$  such that there is no  $t$  with  $t_1 \rightarrow_R^* t$  and  $t_2 \rightarrow_R^* t$ , either add the rule  $t_1 \mapsto t_2$ , or the rule  $t_2 \mapsto t_1$  to  $R$ , such that the extended term rewriting system remains terminating (that's the tricky part and not always possible, i.e. the method may fail here). Set  $R$  to be the extended system and go to 1.

## Exercise: Knuth-Bendix Algorithm

$$R := \{ g(g(z)) \mapsto c \}$$

Is  $R$  terminating?

Is  $R$  confluent?

If not, use the Knuth-Bendix Completion Algorithm to make it confluent.

## Summary: Equational logic

- ▶ The *deduction rules of equational logic*;
- ▶ the notion of a *term rewriting system*  $R$  and associated with it the relations

$$t \rightarrow_R t'$$

$$t \rightarrow_R^* t'$$

$$t \leftrightarrow_R t'$$

$$t \simeq_R t'$$

and the notion of a term in *normal form*;

- ▶ the term rewriting system associated with a system of equations;
- ▶ the *Soundness Theorem* and *Birkhoff's Completeness Theorem* for equational logic; together with they yield the equivalences

$$\forall E \models \forall(t = t') \iff \vdash_E t = t' \iff t \simeq_E t'$$

## Summary: Term rewriting

- ▶ the property of *termination* and some simple techniques for *proving termination*;
- ▶ the property of *confluence*;
- ▶ the *normal form of a term*  $\text{nf}(t)$  w.r.t a confluent and terminating term rewriting system;
- ▶ *the term rewriting system associated with an initial specification*;