

# **Formale Methoden der Informatik**

Course 185.291

WS2013

Lectors: Pichler, Egly

Author: Georg Abenthung

Matr. Nr.: 0726213

Oct. 2013

## **Abstract**

This document are personal notes to the provided Course-Material.

# 1 Computation and Computability

## 1.1 Problems

- Problem has a name
- Infinite set of instances
- Question
- Answer (yes/no on decision question)
- There are different types of problems
- In FMI most problems are decision problems.
- A program should solve a problem in a mechanical way! (Can be solved by a computer or „An algorithm can solve the problem“)
- Alg. should be simple, understandable(shareable), should terminate and should work on ALL possible instances of the problem
- Given a problem P, can we write a program that is an algorithm for P
- Input is a single String or a list of values (def.)
- Our programming language is called **SIMPLE**
- If there doesn't exist a SIMPLE program, there doesn't exist a Java program (or Turing machine program) either

Church Turing Thesis:  
Any algorithm can be programmed in SIMPLE

Goldbach's Conjecture can't be proved. (Every even integer greater than 2 is the sum of two primes) The algorithm to check never terminates until we find a counterexample. If we would have a program, which could tell us, if an algorithm terminates, we could answer the Goldbach Conjecture. BUT: It doesn't exist.

## 1.2 Halting problem (Page 20)

We want to show, that the Halting problem is undecidable, and then show, that the assumption leads to a contradiction.

A program will be applied to a program as input string.

' ...prime

" ...doubleprime

$\Pi''_h$  ...speak „Pi sub h doubleprime“

This prove goes back to Allan Touring.

## 1.3 Page 23

Reachable-Code Problem is similiar to prove than Halting problem. Put some code at the end of the program (Infinite Loop !!?)

## 1.4 Page 24

Decidability implies Semi-Decidability but not vice-versa.

The halting problem is semi-decidable! Semi-Decidability: You'll never know if you haven't calculated far enough, or you'll never reach an end!

## 1.5 Page 26

There are two sources of infinity. It will halt for instances which reach all lines of code, but the algorithm will never stop, when there's dead code.

## 1.6 Page 27

Cantor's enumeration. Give an enumeration to two indefinite sets.

## 1.7 Page 28

compare to Sequential Calculus by Goedel

# 2 Complexity of Problems and Algorithms

## 2.1 Scope of complexity theory

Notation of Papadimitriou will be used in this section

## 2.2 Page 4

Does there exist a path between u and v.

Outer repeat loop is repeated linearly often.

Inner loop also linearly often. At the end it's cubic runtime.

## 2.3 Page 5

The answer to all question: It doesn't really matter in context of complexity theory.

## 2.4 Page 9

the „Big O Notation“ is used in this lecture.  $O(f(n))$

Other notions are also reasonable, but we - in complexity theory - use the Big-O-Notation

Complexity theory doesn't work with **finite** number of Instances for a problem  $\mathcal{P}$ . The Problem must be generalized before.

## 2.5 Page 11 - Notion of O, $\Omega$ , and $\Theta$

the big  $\Omega$  notation is the reverse of the O notation

$O(n^3)$  might be  $0.5n^3 - 17n^2 \dots$

## 2.6 Page 12 - Efficiently Solvable

Can a problem be solved in Polynomial (**P** or **PTIME**) time.

## 2.7 Page 14

The Vertices in a boolean circuit are its „Gates“

## 2.8 Page 19

We don't care if we can solve the Problem  $\mathcal{P}$  in  $O(n)$  or  $O(n^4)$ . The only important thing is that we can solve it in  $\mathbf{P}$

## 2.9 Page 22 - from Boolean Formulas to Circuits

The boolean Input-Gates correspond with the variables used in the formula.

## 2.10 Page 26 - The class NP

The big problem is, that the search space is Exponential. There are  $2^n$  possible assignments.

Although we can often find a solution to a problem  $\mathcal{P}$  with good heuristics.

## 2.11 Page 29

*INSTANCES*( $\mathcal{P}$ )... Boolean Formulas

*CERT*... Models (truth assignments)

## 2.12 Page 30 - Definition of the class NP

see Course material

## 2.13 Page 35

Transform the TSP from an Optimization Problem to a decision problem by introducing a bound.

# 3 Reductions

This is the most important tool in complexity theory. It's used to compare the complexity of two problems.

## 3.1 Page 3 - Basic Idea

Recall the TSP. Compare TSP as a **Decision Problem** (a.k.a. TSP(D)) vs. the TSP as an **optimization Problem**

Construct a:

- Decision problem using an Alg. for Opt.Problem: Does there exist a tour which fits in a specific budget (Budget might be calculated by a TSP-Opt-Alg.)
- Solution for the Opt.Probl using a Alg. for decision Problem: Use Binary search.

### 3.2 Page 4

We have the following setting:

- Problem A: new Problem
- Problem B: old and easy solution available
- Use Problem B to solve the new Problem A
- we say: A is reduced to problem B. (or  $A \leq B$ )

$$A \xrightarrow{R} B$$

We assume, that the reduction  $R$  is feasible in time. We don't want to talk about the complexity of the reduction  $R$ , we want to talk about the complexity of A and B

Another setting:

- Problem A: known as hard
- Problem B: new
- Problem B is also hard. (If there is no efficient method for A there also doesn't exist a simple solution for B)

Complexity. Theory is more interested in the second setting (the negative example).

### 3.3 Page 8

$$\begin{array}{l} \text{R: } A \longrightarrow B \\ x \longmapsto R(x) \\ x \in A \Leftrightarrow R(x) \in B \end{array}$$

### 3.4 Page 9

Recall what CNF (Conjunctive Normal Form on PL0) means. The SAT Problem is NP-Complete (without proof). If only CNF is allowed, the Problem is still NP-Hard. The same if you would allow any arbitrary structure (e.g. DNF).

with 3-SAT we once more reduce the complexity, by reducing the length of the clauses to 3. The problem is still as complex.

The reduction from SAT to 3-SAT is complicated and not covered in this lecture.

### 3.5 Page 10

2-SAT is easily solvable.

#### 3.5.1 Page 10 - Independent Set

Two points should not be adjacent. . .

The problem gets hard, when you try to find a set of a size  $K$

### 3.6 Page 11

2-SAT  $\xrightarrow{R}$  REACHABILITY

### 3.7 Page 12ff - Example of solving the 2-SAT problem

We can construct 6 literals (3 variable  $x_1, x_2, x_3$ )

$$\begin{array}{l} \alpha \rightarrow \beta \equiv \neg\alpha \vee \beta \\ (\alpha \vee \beta) \equiv \neg\alpha \rightarrow \beta \\ (\alpha \vee \beta) \equiv \neg\beta \rightarrow \alpha \end{array}$$

### 3.8 Page 16ff - more complex example

This is a typical way of proving something.

INDEPENDENT SET  $\leq$  3-SAT (Ind.Set at least as hard as 3-SAT)

3-SAT  $\xrightarrow{R}$  INDEPENDENT SET We assume 3-SAT is old and hard.

**Page 18** Choose one literal per clause and try to set it to true. Be sure not to assign true to a variable and its negation.

**Page 19** Divide the equivalence. We start with the direction from right to left. Assume that  $R(x)$  is a positive instance of B and then show, that  $x$  is a positive instance of A.

$K \dots$  number of clauses

**Page 20** find a truth assignment that  $x \in A$  We set those variables true which occur positively in the independent set.

**Page 21** Now we have to prove the other direction.

Showing (ii) is trivial we have chosen  $m$  literals of  $m$  triangles.

Showing (i) is more complicated. Choose an arbitrary pair of vertices and show it's not adjacent. But it's simpler by choosing an indirect proof.

### 3.9 Page 22

the red stuff is important.

ANY(!) two NP-Problems can be reduced to each other. (e.g. the reduction from 3-SAT to INDEPENDENT SET)

$$\mathcal{P}'_{inNP} \leq 3-SAT \leq IND-SET \text{ or also } \mathcal{P}'_{inNP} \leq IND-SET \leq 3-SAT$$

NP is the Class of problems, that can be solved with succinct Certificates. (It only requires polynomial time to find a witness)

So, if a problem is for example reducible to SAT we know, that the Problem is not harder than NP.

$$P \subseteq NP \subseteq PSPACE \subseteq EXPTIME$$

### 3.10 Page 24

$$\begin{array}{l} \mathcal{P}' \leq \mathcal{P} \subseteq P \\ x \xrightarrow{R} R(x) \end{array}$$

### 3.11 Page 26

Informal Proof

### 3.12 Page 27

The formal solution

$$\begin{array}{l} A \xrightarrow{R} B \\ x \mapsto R(x) \\ x \in A \Leftrightarrow R(x) \in B \end{array}$$

## 4 NP-Completeness

### 4.1 Page 3

2-SAT can be solved in P-Time by reducing it to reachability. SAT and 3-SAT are NP-Complete (without Proof here.)

### 4.2 Page 5

This is only a rough proof-sketch, not the whole proof.

### 4.3 Page 6

We are only allowed to assume, that  $\mathcal{P}$  has a polynomially decidable certificate relation. (Else  $\mathcal{P}$  would not be arbitrary.)

### 4.4 Page 8

WE have to show that  $SAT \leq 3 - SAT$   
 $\phi$  and  $\psi$  must not be logically equivalent.

### 4.5 Page 11 - Some NP-Problems

### 4.6 Page 12

Left: Independent set Idea: Take the complement graph (on the right side)

With this idea an reduction can be constructed.

#### 4.6.1 Page 17

$$\begin{array}{l} \text{3-SAT:} \\ \phi = C_1 \wedge \dots \wedge C_n \\ C_i = (l_{i1} \vee l_{i2} \vee l_{i3}) \end{array}$$

### 4.7 Page 18

Whether you make a question more or less restrictive, you can't say beforehand, if solving the problem becomes easier or harder.

## 4.8 Page 20

We have good SAT-Solvers, so sometimes it's good to reduce a problem to SAT first.

## 4.9 Page 26

**Certificate:** Truth assignment is a Certificate in the SAT world. Mapping is a Certificate in the Graph world.

## 4.10 Page 28

Instead of a clause-based formula we could also use a rule-based formula:

$$p_1 \wedge p_2 \wedge p_3 \rightarrow p_4 = \neg p_1 \vee \neg p_2 \vee \neg p_3 \vee p_4$$

## 4.11 Page 30

The other direction.

The more subformulas we have, the harder it's getting to prove, that there is a model.

# 5 Other important complexity classes

## 5.1 Page 6

This implementation is not recursive...

## 5.2 Page 7

Time complexity is one exponentiation higher than its space complexity.

## 5.3 Page 8 - PSPACE

We assume, that PSPACE is a bigger class than NP. (not proved yet.)

## 5.4 Page 10 - Tica Tac Toe (TTT)

No branching for player 1. We only choose one move, which could lead to a winning-strategy.

Exists-For-All alternation is typical for PSPACE:

„Does there exist **A** move for player one, so that for **ALL** moves of P2 ... exists **A** move for P1 so that for **ALL** moves for P2 ...”

Don't check all constellations of P1 in a move. Check them one after another. So the steps of the tree is polynomial bounded. This immediately gives us a PSPACE upperbound.

## 5.5 Page 11

The travelling space is exponentially big, but we are not forced to use it. This is a property of the game not a property of complexity theory.

## 5.6 Page 14

$P \subseteq PTIME$  ...you don't have more time to access the memory more often ...



## 5.7 Page 14

We have two problems, which need PTIME, so we can don't address more than PSPACE memory.

## 5.8 Page 15 - EXPTIME

„2 to the power of a polynom.”

# 6 Turing machines

Some proofs are almost impossible to encode in SIMPLE. some are easier to implement in Turing machines.

## 6.1 Page 4

Transition table of the turing machine.

## 6.2 Page 5

One sided infinitely tape.

( $s, \triangleright, 1001$ ) means (the cursor position, the symbols on the left side of the cursor including the cursor, the alphabet on the right side )

We can store a constant amount of information in the states of the turing machine.

## 6.3 Page 8

the „h” state is used, if we want to produce output. Else the states are „yes” or „no”

## 6.4 Page 10

yields with  $M^k$  means a state can be reached in  $k$  steps.

## 6.5 Page 11

Start and blank symbol are never in the alphabet  $\Sigma$

Definition: If a string is element of a Language L, the the TM should output yes.

## 6.6 Page 18

3 tapes are needed: Input-Tape (RO), Work-Tape (RW), Output-Tape (WO)

**Implement a RO-Tape:** only allow to overwrite with the same symbol.

**Implement a WO Tape:** only allow the cursor to move to the right.

## 6.7 Page 23

with NTM we have a **transition relation** instead of a **transition function**.

## 6.8 Page 25

If we have at least one „YES“-answer, we say, the machine accepts the input.

To say no, **ALL** leavers must have the answer „NO“

# 7 SAT Problems - Preparatory Concepts

## 7.1 Page 5

This type of figure is called **Cactus Plot** In 2002 they solved approx. 40 problems with 2010 Software they solved approx. 170 problems. With hardware of 2010 and problems defined in 2009.

These are real-world problems like hardware optimization of IBM.

We use SAT-solvers because they are good and provided for free in Public Domain.

## 7.2 Page 19

$\nrightarrow = XOR$

## 7.3 Page 21

This kind of translation is called **Structure preserving translation**.

## 7.4 Page 31

The tree shows all ISF's (Immediate subformulas).

## 7.5 Page 33

**Mapping:** We map all variables to truth-values

There are other representations. Here the **iff** representation is chosen by Uwe Egly.

## 7.6 Page 34

$\models \dots$ , „satisfies“

## 7.7 Page 35

The first formula in the example: This is used to prove if the **Modus Ponens** is sound.

**Modus Ponens:** is an inference rule. from two formulas „ $\phi$ “ and „ $\phi \rightarrow \psi$ “ derives „ $\psi$ “

**Modus Tonens:** from two formulas „ $\neg\phi$ ” and „ $\phi \rightarrow \psi$ ” we derive „ $\psi$ ”

Because we can see here  $I$  as an „Eigenvariable” this proof is correct. It is shown **forall**(  $\forall$ ). So we don't have to introduce an additional  $\forall$ -Quantifier

## 7.8 Page 37

An empty clause is an equivalence for **falsum**.

$W$  ... Knowledgebase

$\phi$  ... query

## 7.9 Page 38

We reduce to **Satisfiability** to reuse SAT-Solver. The reduction can be done quick (e.g. with a python script.)

## 7.10 Different normal forms and translation procedures

### 7.11 Page 44 - NNF translation

These are the rules to translate a formula into **NNF**. These rules must be applied in this order.

### 7.12 CNF translation

From the **NNF** we can simply create **CNF** by applying these rules.

### 7.13 Page 48 - Tseitin translation

NFT ... Normal form translation

### 7.14 Page 53

$q$  is labelled twice (with  $l_1$  and  $l_2$ ) because this is a non-optimized version.

### 7.15 Page 63

YOu can only show the existing of models, BUT you don't get the same models for  $\phi$  and  $\delta(\phi)$

SFO ... Subformula occurence

### 7.16 Page 64

NOT logically equivalent, nly satisfiability-equivalence!

## 8 SAT-Problems - Techniques for modern SAT Solvers

### 8.1 Page 2 - Cactus Plot

What are the reasons for that improvements?

These tests have been run on a modern hardware with solvers from 2002-2010. So, the better results are just because of the better implementations of the SAT-solvers.

In modern solvers not only one step backtracking. The solvers jump high above...

## 8.2 Page 5

**DLL** ... „Davis-Loveland-Lodgement” the authors of the first paper (also called DPLL)

## 8.3 Page 6

**CDCL** - Conflict Driven Clause Learning solvers

## 8.4 Page 7

Basic idea of solvers.

## 8.5 Page 14

Red-Arrows ( $\leftarrow$ ) mean: Conflict between these two clauses under the partial assignment. WE are now in a deadlock stage. We can't put away the conflict.

We now apply the first stop criterium. WE now try to backtrack chronologically.

## 8.6 Page 18

After a few backtracks we get an assignment which satisfies all clauses.

**UNIT-Rule** The blue clauses on the slides are unit (c must be 1 to satisfy this clause) the unit rule detects that and sets c to true. The same happens to atom d.

This is called **Boolean Constraint Propagation**.

## 8.7 Page 19

Now we have a model. Because every clause is satisfied

## 8.8 Page 20

The status of the clauses change over time, like shown in this table. After each decision the status change.

## 8.9 Page 22

Horn Clauses: Every clause has at most one positive atom. (But can have many atoms)

## 8.10 Page 23 - Heuristics to select truth assignments

## 8.11 Page 24 - DLIS

Quite an expensive heuristic. Here we go to the approach to satisfy clauses quickly.

## 8.12 Page 25 - Jeroslov-Wang Heuristic

You try to get Unit-clauses quickly, because they are not far away of conflicts. You want to get short clauses here quickly.

## 8.13 Page 26 - Basic SAT Algorithm

PCP Boolean Constraint Propagation

## 8.14 Page 30 -

Each node in the IG is a variable assignment.

dl (Decision Level) 0 for unit clauses ... You don't have a choice. YOU have to chose the correct answer for the remaining variable.

## 8.15 Page 31

(sic!)  $\neg v_1$  should be  $v_1^d$  (dual) here

## 8.16 Page 32

In S2 we have a UNIT-Clause. We can assign the UNIT value.

Keep in mind: on the slides, we often only see parial IG's!

## 8.17 Page 33ff - Example of Implication Graph (IG) GRASP approach

## 8.18 Page 37

We have a conflict graph with the conflict node  $\kappa$

## 8.19 Page 39

It's conflict driven, because we start to learn, after a conflict  $\kappa$  occurred.

## 8.20 Page 43

How can we learn a clause out of this conflict?

decision levels are increased whenever there is made a new decision

The conslict clause is the negated decision clause.

(sic!) at the bottom we should flip  $x_1$  instead to flip  $x_6$

## 8.21 Page 45

By backtrack to decision level 6, the PCB automatically flips  $x_1$  because of the newly learned rule. Which automatically makes  $\neg x_1$  becomes unit.

## 8.22 Page 48

We have a conflict  $\kappa'$  but we don't have a decision ( $x_1$  was unit, not decided.) Now we could try to go back chronologically to an higher dl. But this is not a good idea. WE go back to dl 3 because there we have the chance to change an assignment, else the clause will remain conflicting. The only one which makes the learned clause  $c_11$  non conflicting.

## 8.23 Page 53 - first UIP scheme

## 8.24 Page 54

cut is a partition of the vertice-set. One node of the edge is in S and the other node is in T

**Page 55** These 3 cut edges are the 3 possibilities to cut out the choice-nodes. You have to choose one of them. Each cut brings up a different conflict clause. Which one to choose?

**Page 58** All clauses shown in the previous slides where **Asserting clauses**. Actual SAT-solvers only work with asserting clauses.

## Page 59 - UIP's

**UIP** ... Unique implication point

**Page 60** Choose the cut (see Page 55) according to the first UIP.

**Page 61** only one decision from the highest level will be taken to the second last decision level, to get a unique.

**Page 65** In the res the  $x_5$  has been cancelled like mentioned in the previous slides.

**Page 68 - VSIDS** After adding learned clauses, you don't go back and recalculate, you simply add the new rule. To avoid overflows you simply divide the scores by 2 (or any other number... but 2 is good because it's a bit-shift).

# 9 RECAP: First order logic and theories

## 9.1 Page 6

The ground term is enough to represent the natural Numbers.  $c = 0$  and  $f(c) \dots = ofc$

## 9.2 Page 10 - Semantics of the first order logic.

## 9.3 Page 17

$p(x) \not\equiv p(y)$  because they have a different  $\alpha$ .

## 9.4 Page 23

We want to overload the „ $\models$ ” symbol here.

## 9.5 Page 26 - First Order Theory

We will build a resolver for an „restricted Theory”.

## 9.6 Page 33

$\doteq$  ... see next slide

## 9.7 Page 34f

Read:  $[...] \rightarrow [...]$

Description on the next slide!

## 9.8 Page 36

line 2: the left hand side of the implication must be true

line 3: right hand side of the implication must be false

...

## 9.9 Page 39f

Rule 6 is a protection rule so that you can't use  $\text{cons}(a, b)$  to atoms. (See remarks on the next slide.)

## 9.10 Page 42

Rule 2-5: conjunctions of the left hand side splitted and all assumed true (since rule 1 must hold))

# 10 A decision procedure for equality logic

<https://tuwel.tuwien.ac.at/mod/resource/view.php?id=166539>

## 10.1 Page 7

Identifier ... a variable.

so:  $\text{term} ::= [a|b|c \text{ or } x|y|z \dots]$

## 10.2 Page 9

First we want to get rid of the constants. After that we have extended the language by a new variable for each constant.

## 10.3 Page 12ff

The variables are the nodes of the graph  $G^E$

## 10.4 Page 17

The graph looks the same whether the (In)Equalities are connected by  $\vee$  or  $\wedge$ .

## 10.5 Page 18

The graph is NOT a representation of the formula.

## 10.6 Page 19

Simple circle when you **only** repeat  $v_1$  (see examples on next slide)

**Page 30** The example is an example of **E-unsatisfiability**

## 10.7 Page 32

We throw something away. Make the formula shorter.

Step 4: we replace  $true \wedge smthg = smthg$  etc.

## 10.8 Page 37

After apply the algorithm, redraw the graph  $G^E$  and apply algorithm again.

## 10.9 Page 40

$B_t$  is a conjunction of Transitivity constraints.

## 10.10 Page 43

$G_{NP}^E \dots$  NP stands for „non polar“.

## 10.11 Page 49

Make it chordal: We triangulate the graph.

## 10.12 Page 50

There is a  $P - TIME$  Algorithm to make a graph chordal.

## 10.13 Page 56 - From E-logic to propositional logic

The overall reduction algorithm like explained in detail in the previous slides..

## 10.14 Page 57

Use the POLAR version and not the non-polar version for the analysis. (Improvement, if you want to implement that stuff...).

## 10.15 Page 59

You should be able to answer „Why is a chordal graph better?“



## 11 Eqaulity Logic and Uninterpreted Function Symbols

### 11.1 Page 6

You loose all function ? except **function congruence**.

### 11.2 Page 7

**Why?** When you say, that a property holds for  $\phi'$ , then this property holds also for the special property of „+” (or  $\phi$ ) You prove something on the left for all functions of type **F**. You show a more general theorem, when you replace interpreted by uninterpreted function values.

### 11.3 Page 8

Even simplify the proof search (which is the crucial stuff...)

### 11.4 Page 9

**Remark Definition:** Constants are special function symbols with arity 0.

**Remarks:** Everything except the equality ( $\doteq$ ) is uninterpreted.

### 11.5 Page 11

Loopbound:  $i < 2 \dots$  this is a constant, so you know, when the loop stops.

### 11.6 Page 13

Tse multiplier \* is - still - an interpreted function. We want to replace it with an uninterpreted function.

### 11.7 Page 14

You have to deal with overflow in machine architecture.

The binary multiplication \* is replaced by the binary function  $G$

### 11.8 Page 15

we now discuss, who to get rid of the uninterpreted symbols, we created in the previous slides.

### 11.9 Page 17

FC ...functionality constraints

### 11.10 Page 18

If you have a constant, you associate it with it's value.

### 11.11 Page 21 - Ackermann reduction (AR)) with more than one function symbols

### 11.12 Page 23

This shows how „flat” is computed.

### 11.13 Page 24

Read it as a conjunction over the component-wise equalities.

### 11.14 Page 25

: The  $in \doteq in$  can be simplified to *true*.

We can decide the E-formula by converting it to a propositional formula and decide it then.

### 11.15 Page 31

The red stuff line 4: Put the negation into the formula and translate the implication  $(a \rightarrow b)$  to  $(\neg a \vee b)$

## 12 Deductive Verification of Programs 2013-11-13

**Lector:** Gernot Salzer

### 12.1 Page 5

Building compilers is nowadays much more easy, since there are automatic generators.

### 12.2 Page 8

**Model Checking:** e.g. the program must look in a specific time for user input. This can be checked using Model checking.

### 12.3 Page 9

Program is not adequate: It's difficult to know for a computer, what the customer means.

### 12.4 Page 10

You only take a look at the properties, you are interested in. Not to all the auxiliary variables (e.g. Program for multiplication: you only verify the output, and not that a aux.var. which is used in the program behaves in a specified manner.)

### 12.5 Page 11

**semantics of  $T_{PL}$**  : Normally we don't have a formal semantic of a programming lang. The computer doesn't understand, what the prog. is about. We only have examples in prosa.

## 12.6 Page 14: $T_{PL}$ -Syntax

## 12.7 Page 16

We overload the meaning of  $\mathcal{P}$ ,  $\mathcal{E}$ ,  $\mathcal{V}$  etc. . .

## 12.8 Page 18

Syntactically correctness is a precondition for Semantic correctness.

„A language understanding is a statistical phenomenon.” The first sentence on this slide became a Meta-Semantics since it was introduced by Chomsky.

## 12.9 Page 19

We are interested in the red statements in this lecture.

## 12.10 Page 21

Input-States are an assignment of values to variables.

## 12.11 Page 22

In  $T_{PL}$  we don't have indeterminism, so we don't have several outputs for one input.

## 12.12 Page 23 - Program States

In our language  $T_{PL}$  a prog.-state is simply a mapping from variables to values.

## 12.13 Page 24

Set of configurations: We have those two types of configurations.

$(\mathcal{P}xS)$  . . . not final program state.

$S$  . . . final state

## 12.14 Page 25

there might be no successor configuration. For determinism we have at most one!

## 12.15 Page 26

for the abort-stmt. no transition is defined.

The stuff with the line is an „if . . . then . . . else”

Sequential composition: add an arbitr. Programm after  $p$ .

## 12.16 Page 27

Square-brackets  $([. . .])$  is used to denote the semantics of an expression.

Expressions do not change a state! They only have a result.

## **12.17 Page 29**

Note! „[.]” is overloaded!

e.g. the evaluation of  $[u]$ ... is a Unary function.

the evaluation of  $[b]$ ... is a Binary function.

## **12.18 Page 31**

We have a state  $\sigma$  and we want to evaluate the expression  $[.]$  with the given state.

## **12.19 Page 31**

Syntax is inside  $[.]$  and semantics is outside  $[.]$

## **12.20 Page 33 - Example program run. WHILE**

Sequential composition always means we have a subcomputation.

We have now proven, that the result recording to the semantics and inputs is correct.

## **12.21 Page 34 - Example for ABORT**

## **12.22 Page 35 - Infinite program run.**

You end up with the same configuration than above, that means we have a infinite program.

## **12.23 Page 36**

the star means we have arbitrary many steps for this transition.

# **13 Deductive Verification of Programs 2013-11-18**

## **13.1 Page 3**

We always have to know what exactly we're verifying.

## **13.2 Page 9 - Definition of semantics of a program**

The transition relation is the definition of the semantics of a program.

## **13.3 Page 11 - Difference between $[1]$ and 1**

$[1]$  ... used in the programming language.

1 ... the mathematical semantic of  $[1]$

## **13.4 Page 14**

Note the difference with sequential composition...

THIS is a high level view.

### 13.5 Page 15ff - Proof of if then else

Step 3: apply the semantics definition.

### 13.6 Page 17

The natural Semantic is a little bit easier than Structural operational semantics (SOS). You don't need so much side computations.

### 13.7 Page 22

Only if the program terminates we take care of the output-states  $S_{out}$

**Partially correctness:** We don't care about termination.

For a determination proof you have to show, that the states will decrease. So the lines of codes to be executed must decrease.

Beware: you don't want a operating system to terminate.

### 13.8 Page 26

The four different kinds of calculi are similar.

### 13.9 Page 27

Our quantifiers work only with the states.

### 13.10 Page 29

F-states: all states, that are a model of the formula  $F$  (...all states that make the formula  $F$  true).

Not every set of states can be described by a formula.

### 13.11 Page 30

„whenever“ here means „if“.

Mathematicians don't have the term „terminate“ They say a function is „defined“

### 13.12 Page 31

$\{1\}$ ... this is always true (The semantics of „1“ is true). (Set of all states)

### 13.13 Page 36

Think of your parents. Mother says „no“, father says „yes“  $\rightarrow$  Mother is stronger.

## 14 Deductive Verification of Programs 2013-11-20

gefehlt

- Hoare calculus
- Weakest (liberal) precondition

- Strongest postcondition

## 15 Deductive Verification of Programs 2013-11-25

This is the last lecture of the block 3.

### 15.1 Page 12

„Shift up- and downwards”

### 15.2 Page 16

We can omit the parts we know from the premise.

### 15.3 Page 17

From line 1 to 2: We can omit the existential-operator since  $d$  is not used in the formula.

Block down: Use strongest postcondition to show that left implies the right...

### 15.4 Page 18 - Hoare Calculus

read bottom-up

### 15.5 Page 20

This is the short variant.

### 15.6 Page 22 - Loops

The idea is to unwrap the loop  $i$ -times, so that we don't have a loop anymore.

### 15.7 Page 23

the guess must be verified.

### 15.8 Page 23

Now compute the weakest precondition

### 15.9 Page 29

$c$  must be constant only within the LOOP.

### 15.10 Page 32

the more premises we have, the (potentially) easier it gets to make the proof.

$wht'' \rightarrow wht''' \dots$  the loop will only iterate once more if  $e$  is true.

## 15.11 Page 33

For most cases it holds, that you find  $t \wedge b$  looking at the loop-condition.

## 16 Block 4 - Errors: Know thy enemy - 2013-11-27

Lector: H.Veith

Essentially every program is a single long formula.

### 16.1 Limitations of machine Reasoning

You can write a program which determines if a prog. terminates with the possible answers: „YES”, „NO”, „DON'T KNOW”

### 16.2 System ANalysis by Model Checking

Logical specification is a concrete property like „Termination”. Does the prog. terminate?

The problem is to get the state description. We are talking of a „**State Explosion**”. The graphs are getting extraordinary large. (You may have (e.g.)  $2^{2^{32}}$  nodes in a graph.

## 17 Block 4 - Temporal Logic Model Checking

### 17.1 Page 7

AG ... stands for Always

EF ... possible

AF ... eventually

### 17.2 Page 9 - LTL

LTL ... linear time logic

XXa ... a must hold in the second next state.

XFa ... from the next state x must hold in the future

It's simple to convert LTL into FO, but quite difficult to do it in the other way round.

$G(b \vee Xb)$  ... the holes between b is true can be max. the size of one.

$GFc$  ... at each state  $Fc$  must hold.

### 17.3 Page 19 - Families of CTL's

There are different families of CTL. And variants.

## 18 PDF-File with definitions.

### 18.1 Page 7

$CTL^*$  is a generalization of  $CTL$  and  $LTL$

$Fp$  ... path formulas (you need a path to tell if the statements holds)  
 $AXp$  ... state formula (you can evaluate this formula with only one state)  
 Each state-formula is also a path formula.

## 18.2 Page 10 - Semantics of a formula

Read these rules as a programm which tell you what you can do...

## 18.3 Page 13

in *CTL* only AX, AG AU, AF and EX, EG, EU, EF are allowed, but not (e.g.) EGF. But you can combine those operations. You always have **state formulas**. This is the big advantage of *CTL*

## 18.4 Page 14

In *LTL* you only have **Path formulas**. *LTL* is very useful to find counterexamples.

# 19 Block 4 - Temporal Logic, Model Checking (2.12.2014)

## 19.1 ;model checking is fast...

$K, s \models \phi \dots O(|K| * |\phi|)$

## 19.2 Symbolic Model checking

In the 90's it seemed, that a graph is not the best datastructure to describe a problem. The number of nodes exploded (for a normal size C-Program.)

They decided to use **Binary Decision Diagrams** for Symbolic model checking systems.

# 20 Simulation and abstraction (02.12.2013)

## 20.1 Page 2

**Preorder:**  $A \leq B \wedge B \leq A$  does not imply that  $A = B$  (unlike in total order...)

## 20.2 Page 3

Spoiler ... tries to find a counterexample

Duplicator ... does the same action than the spoiler and tries to answer.

## 20.3 Page 4

The spoiler plays on  $S$ , the duplicator plays on  $I$

The color denotes a value of a property of a specific state. Two nodes with the same color have the same value, but are not the same state.



## 20.4 Page 5

have the same propositions means „have the same color”

## 20.5 Page 10

$a, b, c, d$  are labels and not properties here.

The both graphs are not similar. But they are bisimilar.

## 20.6 Page 12

$M_2$  has a strategy to simulate  $M_1$ . The duplicator just always has to go to the „good”  $b$ . But they are **NOT** bisimilar. The spoiler good go to the Model  $M_2$  and goes to the left  $b$ .

Spoiler goes to the  $b$  on the left. In the next round, the spoiler changes the side of the game ( $M_1$ ). Now the duplicator is on  $M_2$  and on the „bad  $b$ ”.

The duplicator needs to change sides to win.

Formula: For all reachable  $b$ 's there exists a  $d$ . (This does not hold on  $M_2$ )  $\dots AXEXd$  or  $AX(b \rightarrow EXd)$

NB: There must be a „E” in the formula.

## 20.7 Page 15

These two formulas are important for verifying systems.

We want to verify  $M_1$  and construct an  $M_2$  which simulates  $M_1$ . We model check a simplified version of  $M_1$ .

## 20.8 Page 16

Although you can do more in  $N$  it is the smaller one of these graphs.

## 20.9 Page 20

After simplification you get a state-base with three states (instead of all integers.) This minimizes the number of states (now 3) dramatically.

But you lose a lot of information:  $h(7) = h(9) = a_{odd}$  therefore 7 and 19 are equivalent, because they are mapped to the same abstract states.

## 20.10 Page 24

The spoiler plays on  $M$  and the duplicator plays on  $M_r$