

# Revision Lecture

---

0. Introduction.
1. From simple to dependent types.
2. Reduction systems and term rewriting.
3. The  $\lambda$ -Calculus and Implication.
4. The  $\lambda$ -Calculus with products and conjunction.
5. The logical framework.
6. Data types.

# Exams from Previous Years

---

- All exams since 2002/03 in CS-336, CS-M36 are relevant.
- The part of the exams from CS-411 (Critical systems) 2001/02 and 2002/03 referring to topics covered in this lecture are relevant.

# General

---

- **Essentially everything needed should be contained in the notes.**  
(For legal purposes I reserve the right to deviate from that rule.)
- All **courseworks** are **preparation for the exam.**
- Only material covered in the lecture and the coursework should be relevant for the exam.

# Structure of the Exam

---

- As usual, there will be 3 questions out of which you should choose 2.
  - Some students answer all 3 questions.
  - That's only recommended if one notices too late that there is a problem with one question, and chooses to answer another question instead.
  - In most cases, if all 3 questions were answered by a student, the first 2 answered were the best ones – the students didn't have time to answer the 3rd one properly.
  - Then it would have been better, to answer the first 2 questions more thoroughly, or spend time on revising the answers, rather than answering the 3rd question, which then doesn't count.

# Sealing of names

---

- When sealing your names, please make it in such a way that it can be broken easily by
  - folding the part containing your name,
  - and putting the sticker so that it covers one corner of it and hides your name.
- The lecturer has to break the seals, once the exam has been marked, so that he can enter the marks into the spreadsheets.
- This is a lot of work if sealing is done improperly.
- Note that marking itself is done completely blindly.

# Structure of the Exam

---

- **3 Questions.**
- **Question 1** will be **mainly** on the  $\lambda$ -calculus, term rewriting and reduction systems.
- **Question 2** will be **mainly** on the dependent function set, dependent product, disjoint union, proofs in intuitionistic logic and principles of such proofs in Agda.
- **Question 3** will be **mainly** on data types
  - i.e. finite sets, Bool, atom, N
  - and proofs about them in Agda.
- General material (e.g. dependent types in programming) can occur in all questions.

# Structure of the Exam

---

- Agda questions will be in such a way that they can easily be solved by hand.
- Derivations and exact rules are needed only for the  $\lambda$ -calculus incl. products.
- The other rules are important in order to understand dependent type theory; but there will be no explicit questions about them.

# Sect. 0 & 1. Introduction

---

- **4 Principal approaches** for writing verified software.
- **Concept of a type.**
  - Advantages of **typed**, of **untyped** languages.
  - Why are types good for writing correct software?
- **Examples of types** in other languages
  - Scalar types (e.g. Booleans, Integers).
  - Simple compound types (records, arrays).
  - Function types, algebraic types (“data”)
  - Interfaces.

# Introduction (Cont.)

---

- **Examples of dependent types** in programming.
  - Templates (e.g. in C++, Java 5) i.e. parametric types.
  - Digital components.
  - Matrix multiplication.
  - Predicates.
  - Dependent grammars in linguistics.

# 2. Red. Systems and Term Rewriting

---

## ● Notions

- Reduction system,
  - Notions like  $\longrightarrow^*$ ,  $\longleftrightarrow^*$ ,
  - strongly normalising, weakly normalising (difference between these notions),
  - normal form, irreducible,
  - confluence, Church-Rosser.
- Relationship between these notions, examples of systems having and not having those properties.
- Why do we need these notions, what's the problem with having for instance a non strongly normalising reduction system.
- What's the simplest reduction system having certain properties (see cwk).
-

# Term Rewriting Systems

---

- **Too complicated** for the exam:
  - explicit definition of term rewriting system and how to obtain the reduction system
- But you should know examples of term rewriting systems (e.g. terms formed from variables,  $S$ ,  $0$ ,  $+$ ,  $*$ ).

# Sec. 3/4 $\lambda$ -Calculus

---

- The **untyped  $\lambda$ -Calculus**:
  - Bound and free variables (mainly usage),
  - $\alpha$ -,  $\beta$ -conversion (again usage).
- The **typed  $\lambda$ -calculus**:
  - Definition,
  - types,
  - rules,
  - product types,
  - $\eta$ -rule (only some ideas, it wasn't taught in detail this year),
  - $\beta$ -normal form ( $\beta - \eta$ -normal form was not taught this year) (mainly how to compute it),

# Sec. 3/4 $\lambda$ -Calculus

---

- Properties.
  - Non normalisation of the untyped, normalisation of the typed  $\lambda$ -calculus,
  - confluence of the  $\lambda$ -calculus
- derivations (carrying them out).
- Examples, like typing an untyped  $\lambda$ -term, computing normal forms.

# $\lambda$ -Calculus in Agda

---

- Representation of the  $\lambda$ -calculus in Agda.
- The following occurs in various sections:
  - All notations about Agda (`let`, `data`, `sig`, `struct`, `postulate`, `open`,  $\lambda$ , `dep. function type`, `C@_`, `C@A`, ...).
  - Termination checker, and why.
  - $\eta$ -rule not in Agda and why?
  - Logical connectives  $\rightarrow$ ,  $\wedge$ ,  $\vee$ ,  $\forall$ ,  $\exists$ ,  $\neg$ .
  - `False`, `True`, `atom` (both in Agda and in general dependent type theory).
  - Need for dependent types in order to express predicates, quantification.

# 5. The Logical Framework.

---

- **4 kinds of judgements** in dependent type theory:
  - $A : \text{Set}, A = B : \text{Set}, a : A, a = b : A.$
  - Only  $A : \text{Set}, a : A$  are visible in Agda.

# 5. The Logical Framework.

---

- **4 kinds of rules** (formation, introduction, elimination, equality).
- **Constructors** and **canonical elements**.
- **Dependent function type, dependent product.**
- Set vs. Type and higher types.

# Dependent Product/Function Type

---

- Two versions of the **nondependent/dependent product** in Agda
  - using “sig”,
  - using “data”.

# 6. Data Types

---

- **Basic data types** (Booleans, finite sets, disjoint union, natural numbers.)
  - Use in Agda.
- **Atomic formulae** (atom).
- How to represent other **formulae** in type theory and in Agda.
- How to carry out **simple proofs** in type theory (like those done in the coursework).

# Formulae in Agda

---

- Representation of formulas in Agda  $\forall, \exists, \wedge, \vee, \rightarrow$  (all Sect. 5, except for  $\vee$  which is introduced in Sect. 6).
  - Why are we representing e.g.  $\wedge$  by the product?
- Proofs of basic formulae of predicate logic (as in the coursework).
- Simple proofs about them (notions like reflexive, anti-reflexive, transitive, symmetric; proofs of such properties).
- Formulas like equality,  $<$  on Bool,  $\mathbb{N}$  (Sect. 6)
- Computation of equalities on product/disjoint union from the underlying equalities (Sect. 6)
- Principles and limitations of constructive/intuitionistic logic.