A0. Introduction, overview.
A2. Hazard analysis.

B1. Introduction.
B1. The logical framework.
B3. Data types.
General

- According to my data, exam Monday 2/6, 14.00, Dining Room C, Fulton House.
  - Please check.

- **Essentially everything needed should be contained in the notes.** (For legal purposes I reserve the right to deviate from that rule.)

- I will come to the exam in the first half hour (approx.; in case you have questions).

- All three **courseworks** are preparation for the exam.
Structure of the Exam

• **3 Questions.**

• **Question 1** on software engineering aspects.
  
  – Some book work.
    * Essentially “bullets” in the notes.
    * Usually only 3 or 4 required, no need to learn long lists.
  
  – Mainly application of knowledge in examples.

• **Question 2** some general questions about Agda and one verification of critical software in Agda.
Structure of the Exam

- **Question 3** formal treatment of Agda.
  - Simple questions about Agda.
  - One simple derivation using rules.
  - Some derivations in Agda.
Stream A

- A few questions which test concrete knowledge.

- Most software-engineering questions require
  - application of your knowledge to something slightly different (essentially evaluation of techniques, methods)
  - or simple calculations.
• **Definition** of “critical system”, “safety-/mission-/business-critical system”.

• **Areas** of critical systems (only some).

• **Lessons** to be learned from Åsta train crash.
  
  – Some (at least 4; not complete list).
  
  – What is a **root cause**?

  – **Preliminary events, initiating/trigger event, intermediate events** (ameliorating, propagating).

  – **Two aspects of safety critical systems** (software engineering vs. tools for writing safety critical software).

  * What is the **system aspect** of safety critical systems?

• **Tools for writing correct software**: different levels of rigour.
• **Requirements document.**
  - What is a requirements document?
  - Functional/nonfunctional requirements; functional safety requirements/nonfunctional safety requirements; context of operation.
  - Hazard, accidents, incidents/near misses.
    * Definitions.
  - Risk.
    * Definition, calculation.
General System Requirements

- **Dimensions of Dependability.**
  - Reliability, availability, maintainability, system integrity, safety, security, data integrity.
  - No need to learn this list.

- **Definitions** of the notions (reliability, availability, failsafe state).

- **Calculation** of reliability, availability, comparison of the two.

- **Mean time to repair, mean time to failure, mean time between failures.** Calculations.

- **Notions in the area of security** (exposure, vulnerability, attack, threat, control, survivability).
Identification of Safety Requirements, Safety Case

- **Classification** of hazards (w.r.t. what).

- **Risk reduction**
  (Design out hazard; safety devices; interlocks; warning techniques)

- **Safety case** (Definition).

- Why **standards**?
• **Which techniques?**
  - FMEA, FMECA, HAZOP, ETA, FTA.

• **FMEA.**
  - Limitations.
  - Process of FMEA (outline).

• **FMECA.**
  - FMEA + calculation/estimation of probability of occurrence, probability of consequences, criticality.
  - Product = measure for the risk.
• **HAZOP.**
  
  – Outline.
  – Examples of guide words.
    * General use and use for computer based systems.
    * Some example of guide words specific for computer based systems.
      · (Early, late, before, after).

• **ETA.**
  
  – How does it work?
  – Example (including calculation of probabilities).
  – Problems (becomes too big)
Hazard Analysis (Cont.)

- **FTA.**
  - How does it work?
  - Examples (how to draw a circuit).
  - No cut sets.
  - Difference FTA, ETA
    (ETA starts with faults and determines resulting accidents,
    FTA starts with accidents and determines faults resulting...
• **Criteria for choice of programming languages for safety-critical software.**
  - Which factors make languages suitable/unsuitable for writing safety-critical software?
  - Why does one not invent new programming languages, but instead introduces subsets of existing ones?

• **Common reasons for programming errors.**
  - Overview.
Why was Ada developed?

Why is Ada used in critical systems?
- Coming from department of defense.
- High portion of safety critical systems is coming from defense industry.
- Ada as well specially designed for such purposes (e.g. real time systems)
• **Basic principle.** (Subset of Ada; compiles with Ada compilers; annotations which are verified by the SPARK Ada tools).

• What's checked by a **data flow analysis**?
  – Input/output behaviour of parameters, initialization of variables; input parameters are used and output parameters are changed.

• What's achieved by a **information flow analysis**?
  – User specifies interdependencies of variables. These are checked.

• What is the basis for **verification conditions**?
  – Only necessary to know: Pre-and post-conditions in procedures.
Stream B

• Main part of the lecture.

• A few questions which test concrete knowledge.

• Most questions about type theory require
  – derivations using rules (1 subquestion)
  – derivations in Agda (main part of question 2/3),
  – modelling of critical systems.
Stream B (Cont.)

- Similar to what was done in the lecture and in coursework.

- Understanding of basic Agda syntax (minor syntactic errors will not be penalized, as long as it is clear that with the help of a machine one would obtain a syntactically correct proof).

- Simplified examples, since no computers available.
• **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 (eg. Booleans, Integers).
  - Simple compound types (records, arrays).
  - Function types, inductive data types.
  - Interfaces.
B1 Introduction (Cont.)

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

- **Examples of dependent types** in programming.
  - Templates (eg. in C++) ie. parametric types.
  - Matrix multiplication.
  - Predicates.
  - Dependent grammars in linguistics.
• **Simple Derivations using rules.**

• **Basic types** (function type and product) – more how to use them rather than to learn lots of details.
• **4 kinds of rules** (formation, introduction, elimination, equality).

• **Constructors** and **canonical elements**.

• **Dependent function type, dependent product**. (Basic rules and how to use them in Agda).

• **let expressions**.

• **Presuppositions**.

• **Structural rules** (for use in derivations).

• Notion of **Set** vs. **Type** (how to use Set, Type).
B3 Data Types

- **Basic data types** (Booleans, finite sets, disjoint union, natural numbers, vectors of length n, lists).
  - Universes and algebraic data types not treated in this lecture.

- Modelling of the **traffic light example**.

- **Atomic formulae**. atom.

- How to represent other **formulae** in type theory.  
  E.g. \( \forall x : N. A \) becomes \( (x : N) \to A \).
  - No details about constructive logic.
B3 Data Types (Cont.)

- Derivations in the context of \( \mathbb{N} \). (Definition of \( \equiv \), \(<\); simple proofs).

- **Termination checker** (what can it do? what are the limitations).

- Verification of a **circuit** (Coursework 3).