INFO-F-403 – Language theory and compiling
Table of contents

Gilles Geeraerts

December 13, 2013

As a general rule, students are expected to be able to:

- define, using proper formalism, the different notions seen in the course;
- discuss all those notions intuitively;
- provide examples to support the intuitions;
- compare the different notions and explain their relationships to each other.

Theoretical questions will range among the topics given hereunder. Practical questions will be inspired from the problems and exercises solved during the practicals.

Please note that the list of potential questions given hereunder is given to you as a hint and is <u>not exhaustive</u> !

1 Introduction

1. Notions of word, language, alphabet.
2. Notion of compiler.
3. Compiling phases: scanning, parsing, synthesis. Notions of tokens, lexical unit, pattern,...

Examples of possible questions
• Define the different notions of the chapter. • Explain the compiling phases .

2 Regular languages and finite automata

1. Definition of the class of regular languages.
2. Definition of the regular expressions.
3. Definitions of the three different types of finite automata.
4. Definition of the semantics of FAs: configuration, run, language...
5. Equivalences between FAs and REs: proofs that the four models are equivalent.
6. Properties of regular languages.

Examples of possible questions

- Define formally the different notions of the chapter.
- Give examples of the different classes of automata and the notions of configuration, run, language,...
- Give an FA / an RE that recognises a given language.
- Prove the equivalence between FAs and REs.
- Convert an FA into an equivalent RE and vice-versa, applying the systematic techniques seen during the lectures and the practicals.
- Give and explain the closure properties of reg. languages.

3 Lexical analysis

1. Role and place of the scanning during compiling.
2. Extended regular expressions.
3. Using FAs to build scanners.
4. The Lex tool.

Examples of possible questions

- Explain the role and the place of the scanning during compiling.
- Outline the construction of a scanner using FAs.
- Exercises involving extended REs.

4 Grammars

1. Formal definition, notions of derivation (including left-most and right-most) and accepted language.
2. The Chomsky hierarchy.

Examples of possible questions

- Define formally the different notions of the chapter: grammar, derivation, language,...
- Illustrate these notions by means of examples.
- Give a grammar characterising a given language.
- Define the Chomsky hierarchy, and give examples of languages that separate the different classes.

5 Regular grammars

1. Definition of regular grammars.
2. Equivalence between the languages of regular grammars and the regular languages.

Examples of possible questions
• Explain intuitively the equivalence between regular languages and regular grammars.

6 Context-free grammars

1. Definitions of: context-free grammar, derivation tree, ambiguous grammars.
2. Algorithms to clean and simplify grammars: remove useless symbols and left-recursion; factor rules; enforce associativity and priority of the operators.

Examples of possible questions
• Give the formal definitions of the different notions. • Illustrate those notions on examples. • Give the derivation tree of a given word in a given grammar. • Give, explain and justify the algorithms to clean and simplify grammars. • Apply those algorithms on examples.

7 Pushdown automata and properties of context-free languages

1. Definitions of: (deterministic) PDA, acceptance conditions, configuration of a PDA, configuration change, accepted language.
2. Equivalence between PDAs and CFGs.
3. Closure properties of CFGs.

Examples of possible questions
• Give formally the definitions of the chapter. • Illustrate the notions by means of examples. • Give a PDA that accepts a given language. • Characterise the language of a given PDA. • Give and prove the closure properties of CFLs (slide 197).

8 Syntactic analysis (parsing)

1. Role and place of the parsing during compiling.
2. Algorithm to build a non-deterministic PDA from a CFG.
3. Notion of top-down parser: general principle, possible actions, possible conflicts.
4. Notion of bottom-up parser: general principle, possible actions, possible conflicts.

Examples of possible questions

- Explain the role and place of the parsing during compiling.
- Give a non-deterministic PDA that accepts the same language as a given CFG.
- Explain top-down parsing.
- Explain bottom-up parsing.
- Give an accepting run of a top-down parser on a given word.
- Give an accepting run of a bottom-up parser on a given word.

9 LL(k) parsers

1. Notions of First^k and Follow^k.
2. How to compute them.
3. Notion of LL(k) grammar.
4. Characterisation of LL(k) grammars using First^k and Follow^k (slides 250 – 253).
5. Strong LL(k) grammars.
6. LL(1) = strong LL(1).
7. Left-recursive grammars are not LL(k).
8. Notion of LL(1) parser.
9. Notion of strong LL(k) parser.
10. Errors and re-synchronisation.
11. Implementation of a recursive LL(k) parser.

Examples of possible questions

- Define First^k and Follow^k.
- Compute the First^k or Follow^k on a given grammar.
- Define LL(k)/strong LL(k) grammars and compare those notions.
- Give the characterisation of LL(k) grammars in terms of First^k and Follow^k.
- Explain why left-recursive grammars are not LL(k).
- Build an LL(1) parser (give the action table).
- Give the run of an LL(1) parser on a given word, using the action table.
- Build an LL(k) parser (give the action table).
- Explain how errors are detected during LL(k) parsing and how they can be handled.
- Explain how an LL(k) parser can be implemented recursively, and give an example.

10 LR(k) parsers

1. Definition of LR(k) grammars.
2. Canonical LR(0) CFSM: definitions and algorithms
3. LR(0) parser: how to build the tables from the CFSM.
4. LR(0) analysis.
5. LR(k) items: definitions and algorithms.
6. LR(k) parser: how to build the tables.
7. SLR(k) parsers: definitions.
8. Algorithms to build an SLR(k) parser.
9. LALR(k) parsers: definitions.
10. Algorithms to build an LALR(k) parser.
11. Yacc

Examples of possible questions

- Give the formal definitions of the chapter.
- Explain intuitively the definitions, in particular LR(k).
- Give a grammar that is LR(1) but not LR(0) / LALR(1) but not SLR(1) / LR(1) but not SLR(1),...
- Give the algorithms to build the different analysers.
- Build the LR(0)/LR(k) CFSM for a given CFG.
- Build an LR(0)/LR(1)/SLR(1)/LALR(1) parser for a given CFG.
- Give the run of the LR(0)/LR(1)/SLR(1)/LALR(1) parser on a given word.

11 Semantic analysis

1. Role and place of semantic analysis during compiling.
2. Abstract syntax tree.
3. Attributes of identifier, handling types, casting/coercion
4. Semantic actions.
5. Attributed grammar: notions of attributes, synthesised and inherited attributes,...
6. Control flow graph.

Examples of possible questions

- Explain the role and place of semantic analysis during compiling.
- Explain the different tools used during semantic analysis: AST, CFG, semantic actions, attributed grammars.
- Give the CFG of simple given program.
- Give the AST a simple given expression.
- Give an attributed grammar for a given purpose (example: to compute the value of an expression).
- Add semantic actions to a given grammar (example: to print an expression in a given form).

12 Code generation

1. Notion of run-time stack and stack frames to handle recursive functions.
2. Memory management with dynamic memory allocation: stack and heap.
3. Notion of intermediary code.
4. LLVM (see additional slides).
5. Code generation for simple constructs: if, for-loops.

Rem: the slides regarding the P-machine code should *not* be studied.

Examples of possible questions

- Explain how the stack is managed to handle recursive functions.
- Explain how memory is handled in the case of dynamic memory allocation.
- Explain what is intermediary code and why it is useful. Illustrate with LLVM.
- Explain how machine code can be generated for a for-loop, and if,...

13 Turing machines

This topic is *not* covered by this course.