Programming Languages (Principles and Design): Section B, Lecture Review

Dr Paris Yiapanis

room: John Dalton E151
email: p.yiapanis@mmu.ac.uk

Section B Exam Questions

- Three multi-part questions
- You must answer two out of three questions
- Answer all sub-parts from these two questions
- Sample exam questions only for Section B are available on Moodle

What Remains to be Assessed

- Lexical analysis
  - Regular expressions: given a RE what does it mean in natural language, given a natural language description, write a RE
  - Finite automata: NFAs and DFAs, what they are, difference between the two, how you generate them (see revision tutorial)
- Context-free grammars
  - ambiguity: what it is, how to identify it, why it is a problem, how to remove it
  - left recursion: what it is, how to identify it, why it is a problem, how to remove it (the formal process)
- Context-sensitive analysis
  - semantic checking: what it is, identifying examples of semantic errors
  - types: base and compound types, type coercion, structural versus name equivalence
- Machine-independent optimisation
  - Four different types: define and give examples
- Processor architectures, machine-dependent optimisation, cache architectures
  - Discuss how pipelining influences compiler design, including different types of pipelining (e.g. VLIW versus superscalar), and static versus dynamic scheduling

Types of things you might be asked include…

Regular Expressions

- In natural language, what do the following regular expressions mean?
  - letter (letter | digit)*
  - $[+-]?[0-9]^{+}$
  - $a^{*}(a|b)$
  - $(a|b)^{*}ac$

Regular Expressions (cont)

- Write regular expressions that describe the following cases:
  - any sequence of 0s and 1s that ends in 001
  - an identifier that must start with a letter and thereafter can have any sequence of letters, digits and underscores
  - a string consisting of any sequence of letters or digits, followed by “.doc” or “.docx”

Regular Expressions (cont)

- Given a particular regular expression, and a set of input strings, you should be able to identify which would be accepted, and which would not

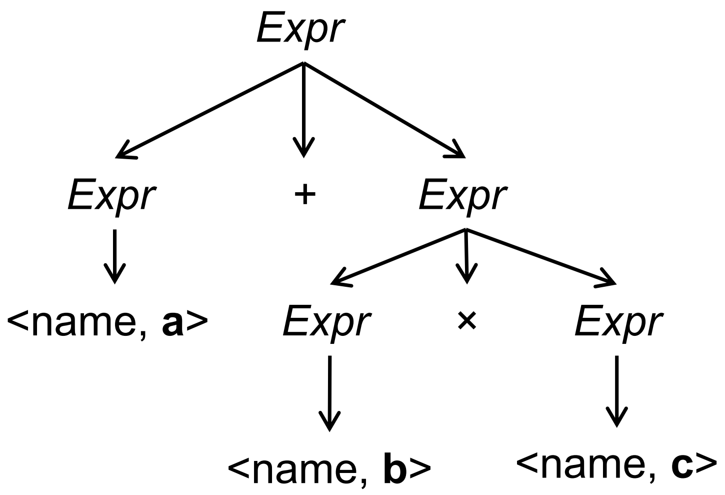
Finite Automata

- Describe the key differences between a NFA and a DFA. Why do we consider both cases?
- For all the REs on slides 5 & 6, you should be able to construct a NFA and a DFA (screencasts week 3).
- You should be able to convert from a NFA to a DFA (screencasts week 3)

An Ambiguous Grammar

1	*Expr*	→	*Expr* + *Expr*
2		|	*Expr* × *Expr*
3		|	(*Expr*)
4		|	name

a + b × c

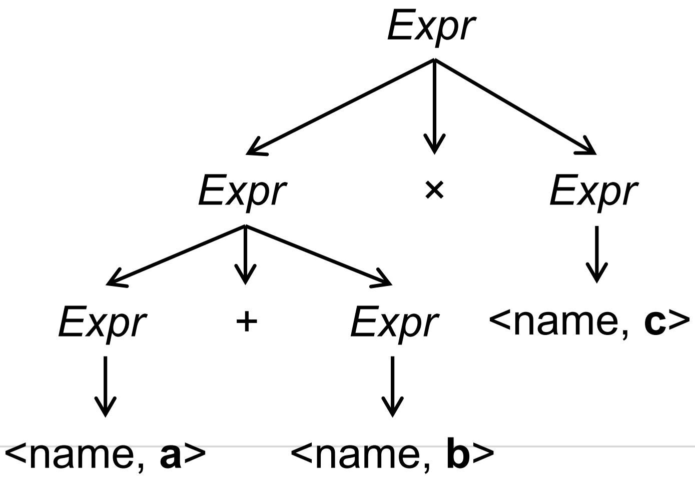


Expr

Expr + *Expr*

<name, **a**> *Expr* × *Expr*

<name, **b**> <name, **c**>



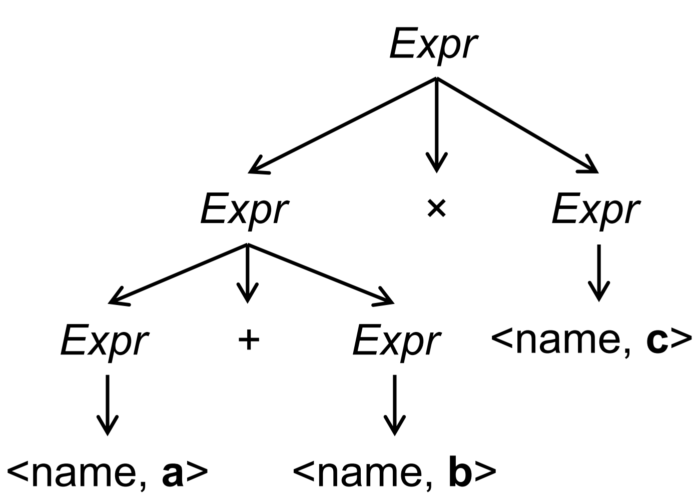
Expr

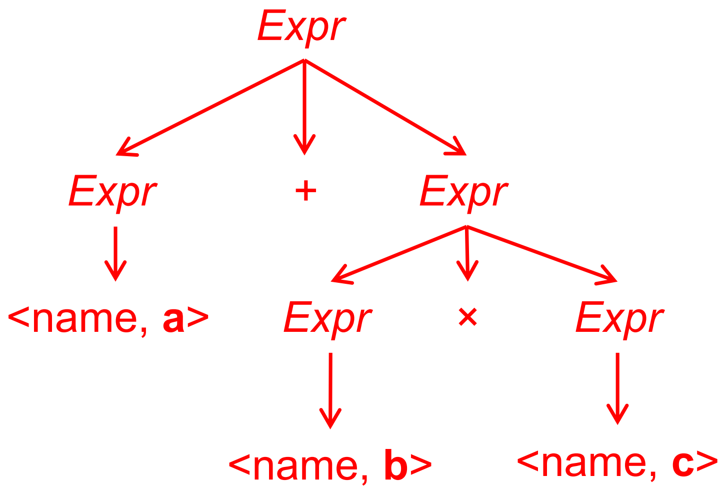
Expr × *Expr*

Expr + *Expr* <name, **c**>

<name, **a**> <name, **b**>

Structure Implies Meaning





Implies
(a + b) × c

Implies
a + (b × c)

Removing Ambiguity

1	*Goal*	→	*Expr*
2	*Expr*	→	*Expr* + *Term*
3		|	*Term*
4	*Term*	→	Term × Factor
5		|	*Factor*
6	*Factor*	→	(*Expr*)
7		|	name

- Now only one possible way to interpret a + b × c
- Ambiguity is removed

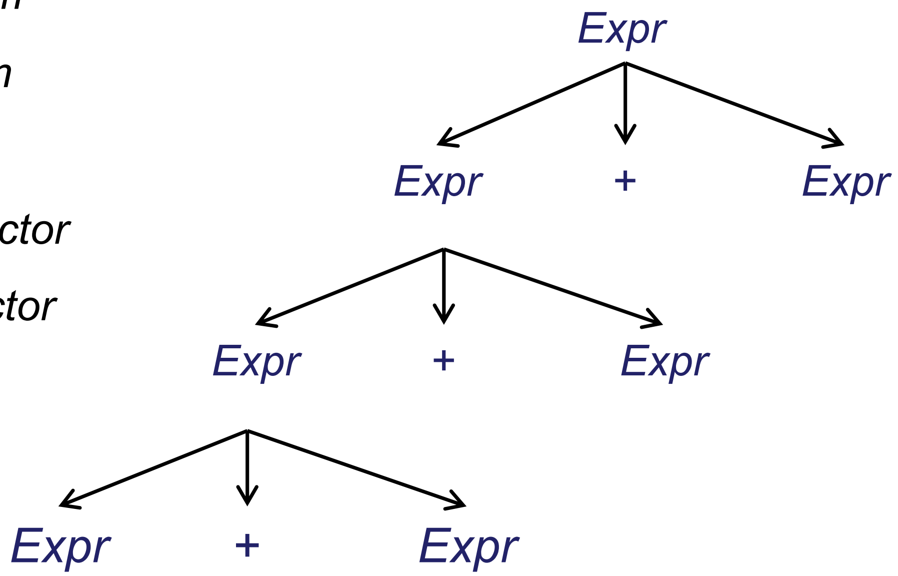
  AND

- Ensure expected operator precedence

The Problem with Left Recursion

1	*Goal*	→	*Expr*
2	*Expr*	→	*Expr* + *Term*
3		|	*Expr* - *Term*
4		|	*Term*
5	*Term*	→	*Term* × *Factor*
6		|	*Term* ÷ *Factor*
7		|	*Factor*
8	*Factor*	→	(*Expr*)
9		|	name

a - b + c



Removing Left Recursion

- Recall:
  - A → Aa | b

  can be replaced by the pair of rules
  - A → bA’ and A’ → aA’ | ε

Semantic Checking

- Given a small-ish block of code:
  - What output would be produced by running it?
  - Explain how a symbol table would be used to check scope of variables (draw a diagram to show symbol table construction)
  - Identify semantic errors in the block

Types

- What is type coercion?
- Given a set of assignment statements, which ones require coercion?
- What is structural equivalence of types?
- What are the differences between base, compound and abstract types?
- What is the difference between static and dynamic type checking (also pros and cons)?

Machine-Independent Optimisation

- Given a block of code, identify the potential(s) for machine-independent optimisation
- Given a type of scalar optimisation, provide a code fragment that illustrates it

Processor Architectures, Machine Dependent Optimisation, Caches

- Describe common architecture features for which compilers can optimise
- Describe the difference between VLIW and superscalar specialisations, and implications for compiler writers
- Explain dynamic versus static scheduling.
  - Explain the importance of optimisation for both cases.

Revision

- Your lecture notes
- Podcasts/screencasts/videos for some topics
- Several *Test Yourself…* topic sheets on Moodle
- Text book
- Online resources
- Previous exams

Answers to Exercises

- Several exercises available to test yourself
  - *Test Yourself…* sheets, sample exam questions
- I will **not** be providing solutions
  - If you have a solution, I'm happy to check it for you
  - If you don't know how to solve a question, tell me where you're stuck and I'll explain it to you

in person

…Reading solutions gives you nothing; doing the exercises means you can do them on the exam.