List of slides

2 0 Course structure
3 Course sessions
4 Learning resources
5 Week by week – lectures and labs
6 Week by week – the project
7 Project marking criteria
8 The examination
9 How much work?
10 Time management
12 1 Formalisation
13 1A It's languages all the way down
14 The Java translation stack
15 Real computers, automatic and manual
16 The essence of computing
17 Programs
18 Is the six-instruction computer sufficient?
19 Are all six instructions necessary?
20 Cheap computing
21 Expensive computing
22 Languages all the way down... with the hardware as a base case
24 1B Utility and power in languages and their processors
25 Reuse encourages layering
26 Internal and external DSLs
27 The DSL maintenance trap
28 The DSL maintenance solution
30 1C What do we mean by *formal*?
31 Formalising mathematics
32 Formalising the notion of computation
33 Programming languages and the Entscheidungsproblem
35 1D Conway's Game of Life, formalised
36 Cellular automata
37 The CGL transition rule
38 Neighbourhoods
39 Translation to Java
40 Some CGL patterns
41 Naïve implementation of CGL using an array
42 A set based implementation without 'edges'
43 Using a visited set to avoid recomputations
44 How is this relevant to programming language design and implementation?
45 Programming languages as transition relations
47 1E The rewrite model: Euclid's Greatest Common Divisor
48 The text of *Elements*
49 The GCD algorithm as a program
50 GCD in Java
51 The fixed-code-and-program-counter interpretation
52 Rewriting – another way to think about program execution
54 Rewriting vs. using a program counter
56 A rewrite evaluation of GCD with input [6, 9]
61 2 Rewriting systems - concepts and terminology
62 2A Transition relations and abstract rewriting
63 Relations
64 Operations over relations
65 Properties of relations R between domain X and codomain Y
66 Homogeneous relations over a set X
67 Closures of homogeneous relations
68 The graph of a relation
69 Transition systems and abstract rewriting systems
70 Normal forms and termination
71 Normal termination and *stuck* transitions
72 Confluence
73 Keeping things simple
75 2B String rewriting

76 Rules for upper casing
77 Rule schemas
78 More subtle rewrites
79 Avoiding implicit equality tests
80 Avoiding contiguous variables
81 Matching, binding and substituting
83 2C Term rewriting
84 Infix syntax
85 Prefix and postfix syntax
86 Internal syntax style
87 Prefix notation and trees
88 Terms
89 The recursive nature of terms
90 Term patterns
91 Matching of terms
92 Notation for the matching operation
93 Pseudo-code for term matching
94 Substitution of terms
95 Pseudo-code for term substitution
96 Some simple rewrites
98 2D Rewriting strategies
99 Subsetting rules and using phased strategies to manage termination
100 Subsetting rules and using phased strategies to manage confluence
101 Expressions naturally need inside out evaluation
102 Simple Java class for binary term constructors
103 Preorder traversal
104 Inorder traversal
105 Postorder traversal
106 Sibling order
107 Incomplete traversals
108 Real systems
110 2E Implementing terms
111 Naïve term representation
112 Better term representation
113 The immense cost of term rewriting
114 DAGs and immutable terms
115 Hash tables of terms: the term is its hash table address
116 Aspects of ART's implementation
117 The CWI ATerm datatype
119 2F Types and operations: the ARTValue system
120 The rise of typing in programming languages
121 Typeful programming
122 Coercion vs conversion
123 Strong and weak typing
124 Static and dynamic typing
125 Name equivalence vs structural equivalence
126 ART's types
127 ART's operations
128 Dynamic type checking of ART values
129 The V3 and V4 ART value systems
130 Atomic types
131 Enumeration types
132 Subrange types
133 Dimensions
134 Polymorphism through overloading and named arguments
135 Polymorphism through flexible arities
136 Polymorphsim through inheritance
137 Polymorphism through parameterised types
139 The emporer's old clothes: https://dl.acm.org/doi/pdf/10.1145/358549.358561
140 3 Structural Operational Semantics
141 3A Conditional term rewriting
142 Configurations
143 Semantics by enumeration
144 A schema for add02

145 Conditional rewriting expressed using operators
146 Built in functions
147 Inference rules
148 Round the clock and recursion
149 Inference rules in detail
150 Abbreviated rules
151 Examples in ART
158 ART's eSOS trace levels
161 3B Recursive inference rules
162 Addition, and compound addition
171 Compact traces
173 3C Expression nesting and congruence rules
174 The three-rule style for binary operators
179 3D The store, program variables, output and simple sequencing
180 Assignment of an integer to a variable
181 Assignment resolution
182 Sequencing
183 Sequenced store operations using previous rules
184 Dereferencing a variable
185 Output
188 3E Control flow
189 if-then-else
192 while-do
196 Other structures
198 3F Big step semantics
204 4 Syntax specification and analysis
205 4A Internal and external syntax
206 Middle end first
207 Thinking about programs and their transformations
208 Syntax for humans
209 Resources for thinking about syntax
210 Taming the syntax monster
211 Backus and Naur and their notation
212 Mathematical games for lexicalisation and parsing
213 Formal grammars
214 Generating sentences
215 The Chomsky hierarchy
216 Derivation steps and context free derivations
217 Derivation trees
218 Properties of derivation trees
219 Unit productions, epsilon-free grammars and uselessness
220 Finite and infinite languages
221 Ambiguity and grammar equivalence
222 Semantics and derivation trees
223 Generators, recognisers and parsers
224 Alphabets, whitespace, and the benefits of lexicalisation
225 Why context free parsing?
226 The traditional Lex/Yacc style external to internal syntax pipeline
227 The 'Holloway' pipeline
228 The 'Holloway' pipeline with specification inputs
230 4 B Limited but easy parsing: Ordered Singleton Backtrack Recursive Descent
231 The pedagogic benefits of OSBRD
232 The graph of leftmost derivation steps
233 An unambiguous transition graph is a tree
234 An ambiguous transition graph is not a tree
235 Unhygenic transition graphs are not even DAGs
236 Parsing strategies
237 Our approach
238 Two external syntaxes for BNF
239 Extended Backus Naur form – EBNF
240 Some EBNF subtleties
241 Lexical matters
242 Some lexical subtleties
243 ART V3 and your project work

244 The OSBRD idea
245 OSBRD for pedagogy, MGLL for production
246 OSBRD example output: `S ::= 'b'|'a' X '@'   X ::= 'x' X | #`
247 OSBRD parse code: `S ::= 'b'|'a' X '@'   X ::= 'x' X | #`
248 Informal parser construction rules
249 OSBRD parse trace: `S ::= 'b'|'a' X '@'   X ::= 'x' X | #`
250 Adding in semantic actions
251 Syntax directed translation output
252 OSBRD semantic functions
253 OSBRD main() and parse() functions
254 Using attributes to transmit information within the tree
255 An attributed example
256 Implementing attributes in OSBRD
258 Tree construction as specialised semantics
259 Representing trees
261 Tree construction and rendering
263 Tree outputs
265 Generated tree functions
266 Skipping tree nodes with promotion operators
267 The ˆ and ˆˆ fold operators
268 Fold under `^` on a terminal
269 Fold under `^` on a nonterminal
270 Fold over `^^` on a nonterminal
272 4C From strings to tokens - lexical analysis
274 4D The space of parsing algorithms
276 4E Generalised Context Free parsing
278 4F Generalised regular and context free lexical analysis
280 5 Grammars, trees and attributes
281 5A Motivation
282 Abstraction in linguistics, science and mathematics
283 Abstraction in programming
284 The idea of abstract syntax
285 Abstraction in syntax
286 Discarding nesting tokens
288 5B Grammar idioms for infix expressions
289 Commutativity
290 Associativity
295 5C Delayed attributes and control flow
297 5D Control flow abstraction
299 5E Formal aspects of attribute grammars
301 5F A synthesis of attribute grammars and term rewriting
302 6 Pragmatics and case studies
303 Further reading on language design and implementation
304 Further reading on machine level topics and software management
305 6A The architecture of language processors
306 It's languages all the way along
307 Why four stages?
308 Do we always need four stages? The single pass compiler
309 Do we always need four stages? The assembler
310 Do we always need four stages? The interpreter
311 Do we need more than four stages? The Just-In-Time compiler
312 Tooling use cases
313 6B The landscape of programming languages
314 How not to design a successful language
315 The perfect language may not exist
316 Programming language facets
317 Selected ideas in programming languages
318 6C A Zoo of specification and implementation technologies
319 L1 Laboratory: Domain specific languages for 3D printing
320 P1 Project: Java FX 3D as an internal DSL
321 Revision topics