Day of the Master

Dr. Vadim Zaytsev

aka @grammarware
Introduction

- Universiteit van Amsterdam (2013–2014)
- Vrije Universiteit Amsterdam (2004–2008)
- Rostov State Transport University (1999–2008)
One year Master of Science programme at UvA

Drifted away from computer science

We teach software construction, evolution, testing, architecture, process, requirements engineering, etc

Programmer in, software engineer out

http://www.software-engineering-amsterdam.nl
FROM CODE-MONKEY...
Roger Delgado as The Master in *The Claws of Axos*
http://www.eyeofhorus.org.uk/images/photo/03pertwee/clawsaxos/master-delgado.jpg
Science
- solves problems
Engineering
- solves problems
Gürbüz Doğan Ekşioğlu, [http://www.gurbuz-de.com/merdivenler-e.html](http://www.gurbuz-de.com/merdivenler-e.html)
Gürbüz Doğan Ekşioglu,
http://markovart.wordpress.com/2014/01/03/surrealism-by-gurbuz-dogan-eksioglu/
What is important in Software Engineering?
What's important?

- Domain analysis
- Educated choices
- Tradeoff awareness
- Human factors
- Communicating with management

(collected during the workshop)
“I’m not a great programmer, I’m just a good programmer with great habits”

–Kent Beck

One year Master of Science programme at UvA

Drifted away from computer science

We teach software construction, evolution, testing, architecture, process, requirements engineering, etc

Programmer in, software engineer out

http://www.software-engineering-amsterdam.nl
Coding Dojo
The Doctor fencing with The Master, *The Sea Devils*, s09e03.
The Doctor fencing with The Master, The Sea Devils, s09e03.
Warm-up!
Recipe (1924):

- take a numerical calculation ($2 \times 2 = 4$)
- replace digits by letters ($A \times A = B$)

Results in:
- cryptarithm

Cryptarithm
- with numbers as meaningful words
- equations as meaningful phrases
Results in
- alphametic

Send Mode Money!

Send More Money!

9567 + 1085 = 10652

Send More Money!
NO GUN NO HUNT!

87 + 908 + 87 = 1082

NO GUN NO HUNT
WILL OBEY DALEK!

\[ \text{6099} + \text{7825} = \text{13924} \]

\[ \text{WILL OBEY DALEK} \]
EXTERMINATE!

+ EXTERMINATE

MONEYMAKING

CRYPTARITHM
Tasks

- Find a solution of an alphametic cryptarithmetic puzzle
- Given a puzzle, find a solution
- Given a puzzle and a solution, check compatibility
- Find a puzzle with only one solution
- Given a desired word, find valid puzzles

Tomchen1989, Cburnett et al, Poker current event.svg, 2011. CC-BY-SA.
do you know the difference between loop and recursion?

Yes

Do you know the difference between loop and recursion?

Yes

Yes

Yes

Yes

Yes

No

No

No

No

No
ds = {*[0..9]};
for (str solution <- {" <N> <O>
              , <G> <U> <N>
              , <N> <O>
              ------------------------
              , <H> <U> <N> <T>"
          | int G <- ds,
          int H <- ds - {G},
          int N <- ds - {G,H},
          int O <- ds - {G,H,N},
          int T <- ds - {G,H,N,O},
          int U <- ds - {G,H,N,O,T},
          G != 0, H != 0, N != 0,
          (O + 10 * N) +
          (N + 10 * U + 100 * G) +
          (O + 10 * N) ==
          (T + 10 * N + 100 * U + 1000 * H)})
println(solution);
```java
str gen(list[str] xs)
{
    keys = sort({x | /str s <- xs, int x <- chars(s)});
    int width = 4*max([size(s) | /str s := xs])+3;
    f = "module Solver
    'import IO;
    'void solveit()
    'ds = {*[0..9]};
    'for (str solution \<- {""
        + intercalate(""
            
        [right(intercalate("","\"<<stringChar(c)>>" | int c <- chars(s)),width) | s <- sx[..-1]
    ]+ ""
        
        right("","-"+"
            
            right(intercalate("","\"<<stringChar(c)>>" | int c <- chars(xs[-1])),width)+"" | 
        
        visited = [];
    'for (k <- keys)
    {
        f += " int <stringChar(k)> \<- ds - {<intercalate(","\n            visited += stringChar(k);
    }
    notzeros = sort({chars(s)[0] | /str s <- xs}):
    f += "t"+intercalate("","\"<<stringChar(c)>>! 0" | c <- notzeros) + ","
        <intercalate(" +\n    ["((<factorise(s)>) | s <- xs[..-1])" == 
        (<factorise(last(xs)))]
    'println(solution);
    }
    'public void main(list[str] args) {solveit();}
    println(f);
    println(f);
    return f;
}
```
HELPING OBSERVATIONS

- Leftmost letters cannot be 0
- The result cannot be too long or too short
- If the result is longer, its left digit is 1
- No puzzle can contain more than 10 different letters
- Brute force solution can be optimised
  - exclude obviously wrong hypotheses
Lessons Leant

- Recursion of known max depth can be rewritten as nested loops
- Harder tasks can be made simple by solving subtasks
- Easier tasks can be inefficiently solved by reuse
- Small differences in requirements matter

(collected during the workshop)
LINES OF CODE?

- Count the number of lines of source code in a file
- Disregarding
  - indentation and whitespace
  - empty lines
  - comments
Solution

- Looping over lines
- Trimming/stripping
- Regular expressions for comments
  - trouble with combinations of //, /* */ and ""
REGULAR LANGUAGES

below context free in the Chomsky hierarchy!

Duncan Rawlinson, Chomsky.jpg, 2004, CC-BY.
Regular Expressions

- Stephen Kleene invented regexps in 195x
- Ken Thompson added them to ed & grep
- POSIX standard since 1993
- PCRE by Philip Hazel (stable release Dec. 2013)

Konrad Jacobs, S. C. Kleene, 1978, MFO.
Archetypal hackers ken (left) and dmr (right).
Regular Expressions

- Stephen Kleene invented regexps in 195x
- Ken Thompson added them to ed & grep
- POSIX standard since 1993
- PCRE by Philip Hazel (stable release Dec. 2013)

Konrad Jacobs, S. C. Kleene, 1978, MFO.
Archetypal hackers ken (left) and dmr (right).
Lessons Leant

- Perfect solutions are sometimes provably impossible
- “Close enough” solutions are useful
- Science: definitive proofs
  Engineering: constant incremental advancements
- Ultimate reuse: find a suitable tool
- Metrics should not be abused (careful reporting)

(collected during the workshop)
GRAMMARS
**3.1 Fundamental Operations**

Compiler Construction

- Dragon Book
- everything you wanted to know about compilers but were afraid to ask
- not needed in practice
LANGUAGE WORKBENCHES

- Rascal
  - http://www.rascal-mpl.org/
- Spoofax
  - http://strategoxt.org/Spoofax/
- ANTLR
  - http://www.antlr.org/
- MetaEdit+
  - http://www.metacase.com/
- MetaProgramming System
  - http://www.jetbrains.com/mps/
- Xtext
  - http://www.eclipse.org/Xtext/

or any old-fashioned parser generator of your choice
module ExprTest

import String;
extend Lexicals;
import ParseTree;
import vis::Expr;

start syntax Expr = "..."

lexical XOp = [+ > [-] > [^] > [/];
Expr getTree(str s) = parse(#start[Expr],s).top;
bool checkTree(str s) = /amb(_!) := getTree(s);

test bool p1() = checkTree(12);
test bool p2() = checkTree(2);
test bool p3() = checkTree(2 // [obj:do()];

rascal>:test
ok

rascal>:showme("2*2+2*2")
ok

rascal>
**Grammar as Parsing Spec**

- Terminals: expected text
- Nonterminals: classes or categories
- Can be combined in a sequence or with choice
- (other workshop-specific fluff)
Need to process an expression language

Don’t write a parser

Write a spec

Concrete syntax def

Let’s start with numbers
Expressions: Numbers

- Grammar is a spec, tests are for the parser
- Numbers do not start with zeros
- There is whitespace around them
We have numbers

Next step:
- operators
- let’s focus on the binary ones

+, -, /, *
**Expressions: Priorities**

expression:
  conditional-or-expression
conditional-or-expression:
  conditional-and-expression
  conditional-or-expression "||" conditional-and-expression
conditional-and-expression:
  inclusive-or-expression
  conditional-and-expression "&&" inclusive-or-expression
inclusive-or-expression:
  exclusive-or-expression
  inclusive-or-expression "|" exclusive-or-expression
exclusive-or-expression:
  and-expression
  exclusive-or-expression "^^" and-expression
and-expression:
  equality-expression
  and-expression "&" equality-expression

Pelf Nyok, Stacked-up river terrapins, 2006, PD.
Expressions: Priorities

equality_expression:
  shift_expression
  equality_expression EQ_OP relational_expression
  equality_expression NE_OP relational_expression

shift_expression:
  additive_expression
  shift_expression LEFT_OP additive_expression
  shift_expression RIGHT_OP additive_expression

additive_expression:
  multiplicative_expression
  additive_expression '+' multiplicative_expression
  additive_expression '-' multiplicative_expression

multiplicative_expression:
  cast_expression
  multiplicative_expression '*' cast_expression
  multiplicative_expression '/' cast_expression
  multiplicative_expression '%' cast_expression

Unknown source (PD?), http://imgur.com/09C5G
**EXPRESSIONS: PRIORITIES**

void LogicalORExpression() : {}
{   LogicalANDExpression() [ "||" LogicalORExpression() ]   }

void LogicalANDExpression() : {}
{   InclusiveORExpression() [ "&&" LogicalANDExpression() ]   }

void InclusiveORExpression() : {}
{   ExclusiveORExpression() [ "|" InclusiveORExpression() ]   }

void ExclusiveORExpression() : {}
{   ANDExpression() [ "^" ExclusiveORExpression() ]   }

void ANDExpression() : {}
{   EqualityExpression() [ "&" ANDExpression() ]   }

void EqualityExpression() : {}
{   RelationalExpression() [ ( "==" | "!=" ) EqualityExpression() ]   }

void RelationalExpression() : {}
{   ShiftExpression() [ ( "<" | "<=" | "=" | ">" | ">=" ) RelationalExpression() ]   }

Murata Seimin, Five Turtles, 1761 and 1837, [http://art.thewalters.org/detail/37228](http://art.thewalters.org/detail/37228)
SIDE STORY: PARSING TECHNIQUES

- Top-down recursive descent parsers: LL(k) etc
  - right recursion is ok, left recursion is deadly
- Bottom-up parsers: CYK, LR(k), LALR, etc
  - right recursion is suboptimal, left recursion is ok
- Top-down parsing, bottom-up lookahead: Earley etc
  - right recursion is ok, left recursion is faster
Expressions: Priorities

Expression:
  Expression1 (AssignmentOperator Expression1)?
Expression1:
  Expression2 Expression1Rest?
Expression1Rest:
  ("?" Expression ":" Expression1)?
Expression2:
  Expression3 Expression2Rest?
Expression2Rest:
  (Infixop Expression3)* | "instanceof" Type
Expression3:
  PrefixOp Expression3
Expression3:
  "(" (Expression | Type) ")" Expression3
Expression3:
  Primary Selector* PostfixOp*

**Expressions: Priorities**

Expression:
  Expression1 (AssignmentOperator Expression1)?

Expression1:
  Expression2 Expression1Rest?

Expression1Rest:
  ("?" Expression ":" Expression1)?

Expression2:
  Expression3 Expression2Rest?

Expression2Rest:
  (Infixop Expression3)* | "instanceof" Type

Expression3:
  PrefixOp Expression3

Expression3:
  ("(" (Expression | Type) ")" Expression3

Expression3:
  Primary Selector* PostfixOp*

**Expressions: Priorities**

```plaintext
def exp:
    'nil' | 'false' | 'true' | number | string | '...' |
    functiondef | prefixexp | tableconstructor |
    exp binop exp | unop exp;

binop
    : '+' | '-' | '*' | '/' | '^' | '%' | '..' |
      '<' | '<=' | '>' | '>=' | '==' | '~=' |
      'and' | 'or';

unop
    : '-' | 'not' | '#';
```

---

Kazunori Sakamoto, Lua 5.2 grammar in ANTLR4,
https://github.com/antlr/grammars-v4/blob/master/lua/Lua.g4
**Expressions: Interpreter**

- Recogniser tells us which program is correct
- Parser builds a parse tree
- Some want a semi-auto-derived abstract syntax tree
- We need smth to walk that tree and execute it
  - e.g., perform calculation
  - e.g., check for correctness
Expressions: Variables?

- how to introduce variables into a software language?
- many different ways
  - each one has implications
- very hard to do w/o background

\[(\lambda x . 2 + x) \ 2\]

\[2 + x \ \text{where } x=2\]

\{x=2;\text{return } 2 +x\}
Lessons leant

- Full TDD: start with failing tests, adapt, repeat
- [1-9][0-9]*|[0] is uglier than [0-9]+*
- Comments should not show up explicitly in the grammar
- Separate tree validation function is useful
- Parsing can be ambiguous
- Several ways to specify/encode priorities

(collected during the workshop)
Summary

- Software engineering vs computer science
- Alphametic cryptarithmetic
- Calculating lines of code in a software system
- Software language engineering
- Mining software repositories (no time)

http://grammarware.net/talks/#DayOfTheMaster2014
Coding Dojo Feedback

- What went well?
- What did not?
- What did we learn?
Feedback

- Hacking is fun. Moar hacking!
- Brief SLE intro appreciated
- Background info feels CS, not SE

(collected during the workshop)
SOFTWARE ENGINEERING

- One year Master of Science programme at UvA
- Skills covered:
  - programming
  - critical thinking
  - solving unsolvable
- Skills uncovered:
  - reading scientific papers
  - collaborating in bigger projects

http://www.software-engineering-amsterdam.nl

vadim@grammarware.net
- Designosaur by Archy Studio
- Heuristica by Andrej Panov
- Matt Smith Doctor Who by thejamjar
- GNU Typewriter by Lukasz Komsta
- Dalek by K-Type
- Doctor Who screenshots by BBC
- logo by Vadim Zaytsev
- logo by Tobias Baanders
- Everything used strictly for educational purposes
- Yours, @grammarware