Automation for Exercises in Computer Science and Mathematics

Johannes Waldmann, HTWK Leipzig

HTWK, 12 November 2018

Exercise: Greatest Common Divisor

- exercise:
  - instance: numbers $a, b \in \mathbb{Z}$, e.g., $a = 30, b = 17$
  - solution: numbers $c, d \in \mathbb{Z}$ such that $ac + bd$ divides both a and b
- example submission: `(-1, 2)`, response:

```
a * c + b * d = 4
4 does not divide 30 (remainder is 2)
```

- solution can be obtained via extended Euclid's algorithm (but system only checks the result, not how it was obtained)

Exercise Instance Generator

- teacher sets parameters for generator

```
Param
  { lower = 10 , upper = 50
  , max_divisor = 10
  }
```

- then system generates a fresh problem instance per student

Exercise: RSA public key decryption

- problem instance: public key (e, m) and encrypted message c, e.g., $e = 7, m = 55, c = 9$
- problem solution: plaintext message p with $p^e \equiv c \pmod m$
- systematic solution:
  - $n = \phi(m) = \phi(5 \cdot 11) = 4 \cdot 10 = 40$,
  - $\gcd(e, n) = \gcd(7, 40) = 1 = 7 \cdot 23 - 40 \cdot 4$,
  - $c^{23} \equiv p^{7 \cdot 23} \equiv p^1 \pmod{55}$.
  - $c^{23} \equiv 14 \pmod{55}$

Exercise: Electrical Circuits

- instance: circuit descrition with holes

```
Circuit { ground = Node 0 , output = N
  , components = [ ( Node 1 , Voltage
    , ( Node 1, Resistor _, Node 2)
    , ( Node 2, Capacitor _, Node 0)
    , ( Node 2, Inductor _, Node 0) ]
```

  and input/output behaviour for specific input functions (impulse, sine wave, . . .)
- solution: complete circuit description

```
    , ( Node 2, Capacitor (1.5 Farad),
```

  that realizes given behaviour close enough

More Exercises

- graph "theory", discrete mathematics:
  - instance: graph G, solution: Hamiltonian Circuit in G
  - instance: graph G, number k, solution: conflict-free k-colouring of G
- logic:
  - instance: propositional logic formula in CNF solution: a satisfying assignment
  - instance: formula in 1st order predicate logic solution: a model of the formula
- principles of programming languages
  - static typing, also polymorphic

Leipzig autotool — General Design

for each type of exercise:

- types: Config, Instance, Solution (each with pretty-printer, parser, API doc)
- functions:
  - grade: Instance × Solution → Bool
  - → Bool × Text
  - describe: Instance → Text
  - initial: Instance → Solution
  - generate: Config × Seed → Instance

Leipzig autotool — Components

- collection of exercise types as (stateless) semantics server (XML-RPC)
- plugin for Olat LMS (learning management system)
- stand-alone autotool LMS with
  - data base (problems, students, grades,. . .)
  - web front-end (for student, for teacher, . . .
  - . . . display highscores: small/early solutions)
- since ≈ 2000, open-source (GPL), Haskell, ≈ 1500 modules, ≈ 15 MB source `https://gitlab.imn.htwk-leipzig.de/autotool/all0`

Leipzig autotool — Applications

at HTWK Leipzig, IMN, since 2003, in lectures on

- Modellierung (discrete mathematics and logic)
- Algorithms and Data Structures
- Automata and Formal Languages
- Advanced (i.e., Functional) Programming
- Artificial Intelligence
- Principles of Programming Languages
- Theory of Computation
- Constraint Programming

Experience - Students, Teachers

- autotool is: always available, always correct, always patient
- teaching/grading assistant is: available for few hours a week only (if at all – staff costs money, which we generally don't have)
- autotool homework exercises prepare students for discussing "real homework" (that is, proofs) in classes

Experience - Implementation

- each exercise type is a domain specific language (concrete syntax, abstract syntax, semantics)
- *implementation* of the grading algorithm (= semantics) is always the easiest part
- the hard part is the *design*
  - what type of exercise helps the student to understand a specific concept?
  - how can we write the instance generator?

Design Goals for Exercises

- grading:
  - should give reasonable explanation for wrong submissions (not just "it's wrong")
  - without giving away the correct solution
- generator:
  - each instance: non-trivial, but manageable,
  - set of instances:
    sufficiently distinct, but of similar difficulty
- concrete syntax:
  - Haskell syntax for tuples, lists, records
  - except: (model) programming languages

Design Principles for Exercises

- basic approach: verify property of an object
  example: any NP complete problem, e.g., SAT
- but this does not check whether the student used a certain algorithm to construct this object
- several exercise types implement non-deterministic algorithms (= inference systems)
  student has to find an execution path (inference tree, proof), examples:
  - Resolution (derive empty clause)
  - Hilbert style deduction (derive formula)
  - (balanced) search tree operations

Example: Algorithms on Search Trees

- instance: AVL trees s, t, pattern p, e.g.,
  `[Insert 92,*,*,*,*,Insert 51,*,Delete 38]`
  solution: sequence q of operations that matches p and transforms s to t
- this exercise is not to implement operations, but to give correct (black-box) implementation so that students can explore their properties
- underlying design principle: *sudoku*,
  that is, create "holes" that students have to fill in

Design Principle: AST Sudoku

- start from any exercise type with
  grade: Instance $\times$ Solution $\to$ Bool
- build generator that produces correct pairs
- Instance $\in \mathsf{Term}(\Sigma)$, Solution $\in \mathsf{Term}(\Gamma)$,
  from Term to Pattern: introduce (several)
  - variables for subtrees
  - variables for function symbols
- "sudoku" variant of this exercise:
  - instance: $(p_i, p_s) \in \mathsf{Pat}(\Sigma) \times \mathsf{Pat}(\Gamma)$
  - solution: a correct instance of (p_i, p_s)
- unlike Sudoku, solution is not necessarily unique

Sounds Great - I Want This!

- autotool is free software (GPL):
  you can download, compile, install, use!
  source/instruction: `https://gitlab.imn.htwk-leipzig.de/autotool/all0`
- TODO (contributions welcome)
  - translation (most exercises German-only, some English-only, some have both texts)
  - more exercise types (requires: 1. design skills, 2. Haskell skills)
  - integration with other LMS (learning management systems)