Universiteit Utrecht

[Faculty of **Science**
Information and Computing Sciences]

Holmes for Haskell
Keeping your Haskell courses free of software plagiarists

Jurriaan Hage Brian Vermeer Gerben Verburg

Department of Information and Computing Sciences, Universiteit Utrecht
J.Hage@uu.nl

April 3, 2013

The situation at University of Utrecht

- FP is mandatory, currently over 200 students per year
- Plagiarism is known to occur in courses involving C# (formerly Java)
- Self-developed Marble works well for Java and C#, but not for Haskell
  - Marble exploits their many syntactic redundancies
- So we developed our own plagiarism tool Holmes

Why can something like Holmes work?

- Observation 1: A student who plagiarises suffers from lack of time, lack of programming skills or both.
- Observation 2: A modification of a submission must be reasonable, at the very least human-readable.
- Observation 3: One hint of plagiarism is enough.

Holmes main characteristics

- All of Haskell

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function
- Performs a simple form of dead-code removal
  - More on this later

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function
- Performs a simple form of dead-code removal
  - More on this later
- Per module and per submission comparison

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function
- Performs a simple form of dead-code removal
  - More on this later
- Per module and per submission comparison
- Performs reasonably well

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function
- Performs a simple form of dead-code removal
  - More on this later
- Per module and per submission comparison
- Performs reasonably well
- Can deal with substantial reordering of code

Holmes main characteristics

- All of Haskell
- Support for comparing against previous incarnations
- Able to deal with template code
  - Per module or function
- Performs a simple form of dead-code removal
  - More on this later
- Per module and per submission comparison
- Performs reasonably well
- Can deal with substantial reordering of code
- Holmes is run locally

What about MOSS?

- Developed by Alexander Aiken and others
- Well-known, well-used
- Supports Haskell and many other languages
- Detemplating for free
- No dead-code removal
- Sensitive to (considerable amounts of) renaming and reordering
  - See Hage, Rademaker and Van Vugt, CSERC 2011
- Source code must be transmitted over the Web
  - MOSS deletes results after 14 days
- Results are quite good

Dead code removal

- Lecturer provides "starting points".
- Only code reachable from starting point will be retained.
- Specification examples `Main.*` and `*.main`

```
useful = .....
spurious = ....
cleverlyHidden = ...

main =
  let
    f = (spurious, useful)
    g = cleverlyHidden
    h = useful
  in const h g
```

Implemented heuristics

- Implementation by Brian Vermeer had more than a dozen
- In the final version only five are used for comparison
  - fingerprinting
    - Taken from MOSS, information theory, generic
  - tokenstream
    - As in Marble
  - indegree signature of top-level functions (compared in three different ways)

Token stream

```
module Token where
f y = 3
main :: Int -> IO()
main x = do
           putStrLn (f (x + x))
           return ()

X X = I X X = do X ( X ( X 0 X ) ) X ()
```

Sorting the functions: by arity, number of tokens, alphabetically
Haskell's Diff library used for comparing the token streams

How to use Holmes

- holmes-prepare computes the metric data for each submission in the current year.
- holmes-compare compares these to metric data for former years, and amongst themselves for the current year.
  - The distinction is made for the system to scale up
- And generates a prioritized list of possible cases of plagiarism.

Sample output

```
fingerprints; tokens; ind1; ind2; ind3; sub VS sub;
015; 067; 076; 048; 079; 2007/xx-yy VS 2007/zz1;
019; 067; 052; 034; 069; 2007/xx-yy VS 2001-hugs/zz2;
026; 064; 068; 068; 079; 2007/xx-yy VS 2004/zz3;
```

- Import into Excel for easy manipulation
- Scores between 0-100, with 100 for "very similar"

Empirical validation

- Sensitivity analysis: how do various classes of code changes/refactorings affect scores?
- Apply Holmes to a large corpus of student programs

Sensitivity Analysis

Single refactorings	
Name	*Description*
nc	changed identifier names
tc	translated comments from Dutch to English
rl	changed the order of the function declarations
rw	simple transformations like `where` to `let - in`
trc	declared a trace function similar to the Debug module and let all functions call trace
cp	move single used functions to local scope
un	declared a unit test function that calls all functions declared in the module

Sensitivity results

original VS ...	tks	in1	in2	in3	fps
nc	100	100	100	100	68
bogus	3	12	4	12	0
trc	85	92	46	92	68
tc	100	100	100	100	100
rl	100	100	100	100	91
rw	87	85	86	94	78
compact	86	94	58	94	99
unit	91	61	60	80	86
nc_rw	87	85	86	94	53
nc_rw_tc	87	85	86	94	53
nc_rw_tc_cp	77	83	62	89	53
nc_rw_tc_cp_trc	74	84	67	92	42
nc_rw_tc_cp_trc_un	68	58	58	81	37
nc_rw_tc_cp_trc_un_rl	68	58	58	81	36

Applying Holmes to real life data

- On all submissions submitted to the FP course 2001-2011
- Organised per assignment/incarnation/submission
- No detemplating, `*.*`, only submission level

Some statistics

total submissions	2122 (out of 2150)
submission to submission comparisons	230688
different assignments	18
total incarnations	36
max repeats for an assignment	7
total students	1042
max assignment for any student	11

Corpus

Assignment name	incarnations	submissions
fp-afschrift	1	65
fp-afschriftgui-ghc	1	62
fp-agenda	1	78
fp-beeldverwerking-ghc	1	59
fp-creditcardvalidation	1	93
fp-fpcal	1	68
fp-fql	6	420
fp-getallen	1	95
fp-html	1	68
fp-kalender	1	6
fp-mastermind	2	156
fp-propositielogica	7	380
fp-river	1	70
fp-rocks	1	70
fp-soccer	2	52
fp-spreadsheet	1	5
fp-turtlegraphics	4	163
fp-wiki	1	74
fp-wisselkoers	1	163
fp-wxcal	1	52

Results

- 63 cases of clear cut plagiarism, 3 cases of fraud
- 12 additional cases that were less clear cut
- 27 cases of plagiarism by copying from a previous incarnation
- Only seven cases had a lot of identical code
  - Refactoring/rework have been performed otherwise
- Tokenstream and fingerprinting each have something to contribute
  - tokenstream counters identifier translation and moving code around
  - fingerprinting works well for (small) partial exact overlaps (resubmissions)
- Indegree signatures somewhat erratic still

Want to see an example?

See the final pages of the paper.

Summary

- Results are very promising, even without file-to-file submissions
- With better suited assignments, Holmes is likely to do better
  - give 'em enough rope!
  - one starting point
  - template annotations if necessary

Recent and Future Work

- Make Holmes public on Hackage
- Compare with MOSS experimentally
- Am I your Turtle or are you my Moriarty?
- Dealing with weak points of Holmes
  - improvements to heuristics
  - "many small changes issue"
  - diff the control-flow graphs
    - What we have does not scale...