Towards Formal Proof Metrics

David Aspinall, University of Edinburgh
Cezary Kaliszyk, University of Innsbruck

FASE, 7th April 2016

Outline

Motivations

From Software Engineering to Proof Engineering

Proof developments, abstractly

Metrics for formal proof

Experiment and analysis

Conclusions

Interactive therorem proving

Interactive Theorem Proving (ITP) is now middle aged.

Can produce large formal proof developments: establish complex properties in mathematics or software verification, indefeasibly.

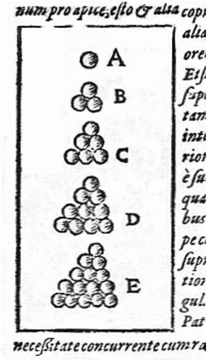
Large developments consist of many lines of “source code”, a form of program, developed interactively (think read-eval-print, scripting languges, rich IDEs).

A number of competing proof languages exist, each unique to a system. Popular examples: HOL4, HOL Light, Isabelle, Mizar, Coq.

Heroic formal proofs

2003-2014: Thomas Hales led the `Flyspeck` project to formalise his proof of the **Kepler Conjecture** in HOL Light.

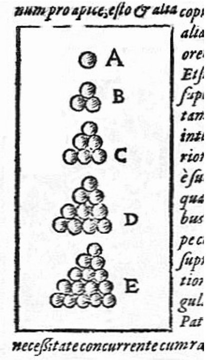
Took circa 20 person years, the text part of the proof consists of **14,185 HOL Light theorems**.



Heroic formal proofs

2003-2014: Thomas Hales led the `Flyspeck` project to formalise his proof of the **Kepler Conjecture** in HOL Light.

Took circa 20 person years, the text part of the proof consists of **14,185 HOL Light theorems**.



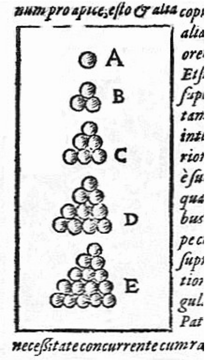

```
let AAUHTVE=prove(`!x:real^3  (V:real^3->bool) (E:(real^3->bool)->bool) p.
FAN(x,V,E)/\ p = ( \ t. res (t x V E ) (d1_fan (x,V,E)) ) ==>
FINITE (d_fan (x,V,E))
/\ (p e_fan) permutes (d_fan (x,V,E))
/\ (p n_fan) permutes (d_fan (x,V,E))
/\ (p f1_fan) permutes (d_fan (x,V,E))
/\(p e_fan) o (p n_fan) o (p f1_fan)= I
/\ (!a. a IN d1_fan(x,V,E)==> (e_fan x V E)  (e_fan x V E a) =a)
/\ (!a. a IN d1_fan(x,V,E) ==> ~(e_fan x V E a=a ))
/\ (!a. a IN d1_fan(x,V,E) ==> ~(f1_fan x V E a=a ))
/\ (!k:num l:num a. a IN d1_fan(x,V,E) ==>
(((n_fan x V E) POWER k) o (e_fan x V E)) a= ((e_fan x V E)o((n_fan x V E) POWER l)) a==> (n_fan x V E POWER l) a=a)

/\ (!n:num a. a IN d1_fan(x,V,E) ==> ~(e_fan x V E a =(n_fan x V E POWER n) a))`,
```

Heroic formal proofs

2003-2014: Thomas Hales led the `Flyspeck` project to formalise his proof of the **Kepler Conjecture** in HOL Light.

Took circa 20 person years, the text part of the proof consists of **14,185 HOL Light theorems**.



Project Cost Calculator

Include

Average Salary (per year)
$ 55000 .00

Codebase Size
344,091 lines

Estimated Effort
89 person-years

Estimated Cost
$ 4,902,018 *

*Using the Basic COCOMO Model

Estimate seems way too high?

Ohloh scans *all files* at any given code location to calculate the cost estimate.

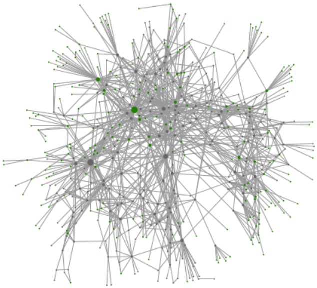
Ohloh lets you exclude files and direc-tories from this calculation on the Code Locations page. You can get a more realistic estimate by excluding:

- External dependencies or libraries
- Non-code files

Heroic formal proofs II

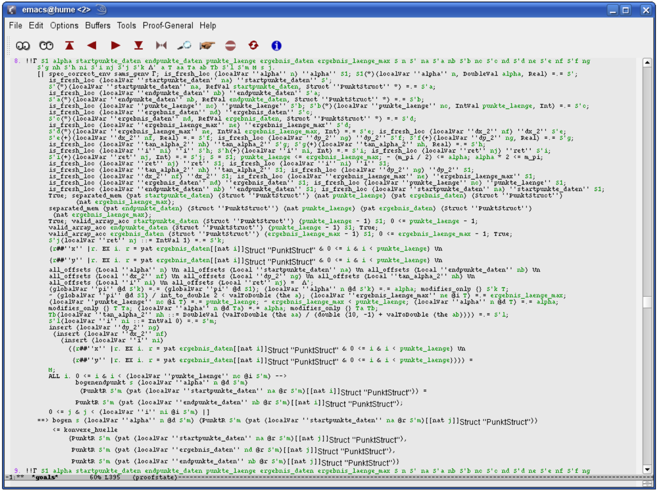
2009: Gerwin Klein and team announced the formal verification of the **seL4 Microkernel** in Isabelle.

Kernel is 8700 lines of C, 600 assembler. Size of proof is 200k lines. Verification effort 12 py (+12 py for tooling).



	Haskell/C		Isabelle	Invar-	Proof	
	pm	kloc	kloc	iants	py	klop
abst.	4	—	4.9	~ 75	8	110
exec.	24	5.7	13	~ 80	3	55
impl.	2	8.7	15	0		

Sample verification condition statement



Why are these efforts so heroic?

Large formal proofs in mathematics or software verification are inherently challenging:

- big formal-informal gap
- statements may be overly complex
- proofs hard to understand
- pioneering, needs new techniques
- ...
- **engineering at scale is poorly supported**

Outline

- Motivations
- From Software Engineering to Proof Engineering
- Proof developments, abstractly
- Metrics for formal proof
- Experiment and analysis
- Conclusions

Proof Engineering

> *Many of the issues faced [. . .] are similar to those in software engineering: there is the matter of merely browsing, understanding [. . .]; there are dependencies between lemmas, definitions, theories, and other proof artefacts that are similar to dependencies between classes, objects, modules, and functions; [. . .] there is the issue of refactoring existing proofs either for better maintainability or readability, or even for more generality and additional purposes; and there are questions of architecture, design, and modularity in proofs as well as code.*

*Gerwin Klein, **Proof Engineering Considered Essential**, International Symposium on Formal Methods, 2014.*

Open questions in Proof Engineering

- How should a large proof be broken into modules?
- How can we tell if a proof is well-structured?
- How can we improve its understandability?
- ...or maintainability?
- If one part is changed, how much will break?

To approach these questions, we need to understand what works better and what doesn't. To do that, we need to measure.

What can we measure?

Dream future: we can find measurements that help *predict* effects that can be empirically validated.

Possible future: we define *metrics* which correspond to or correlate with observed effects.

Current point: we start with *candidate metrics* with sensible properties and that can be tested against current data, refuted and refined.

Potential starting inspiration: **software metrics**.

OO Design: Chidamber and Kamerer, 1994

Rather dated and much debated!
But (AFAWK) no single newer generic better suite.

Six functions, each a metric on a class in a design:

1. Size of the class
2. Role within design: depth in hierarchy
3. Role within design: oversight in hierarchy
4. Responsibility of methods in class
5. Lack of Cohesion within modules
6. Coupling between modules

Generally: interval scale, higher → concerning.

S.R. Chidamber and C.F. Kemerer. *A metrics suite for object oriented design*. IEEE Trans. on Software Engineering 20.6, 1994.

A loose analogy: OOP vs Proof Languages

Proof languages use in-the-large modular structure.

We can match this against OOD structure.

OOP	**Formal proof**
class	proof module
class inheritance	proof module import
instance variable	declaration of a type or constant
method	theorem
method type	theorem statement
method body	theorem proof

Outline

Motivations

From Software Engineering to Proof Engineering

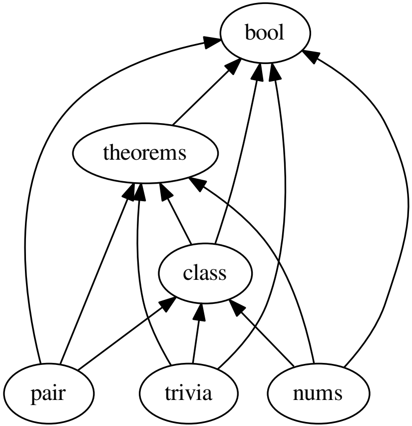
Proof developments, abstractly

Metrics for formal proof

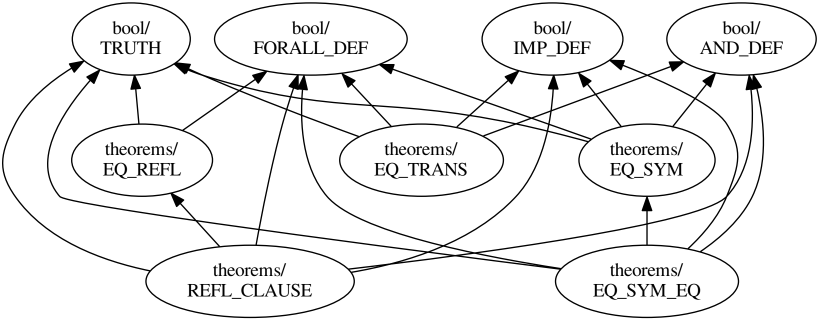
Experiment and analysis

Conclusions

Module dependencies



Theorem dependencies



Features

$$\forall \mathtt{V}.\ \mathtt{packing}\ \mathtt{V} \implies (\exists \mathtt{c}.\ \forall \mathtt{r}.\ \&\ 1 \le \mathtt{r} \implies \&(\mathtt{CARD}(\mathtt{V\ INTER\ ball}(\mathtt{vec}\ 0,\mathtt{r})))) \le \mathtt{pi} * \mathtt{r\ pow}\ 3\ /\ \mathtt{sqrt}(\&18) + \mathtt{c} * \mathtt{r\ pow}\ 2)$$

fea(kepler_conjecture) =
{packing, sqrt, ball, pi, BIT0, BIT1, NUMERAL, 0,
real_add, real_div, real_le, real_mul, real_of_num
real_pow, CARD, INTER, 3, cart, num, prod, real}.

Proof Development Metamodel

- **Module**: (M, T), M is a name, T a set of names t
- **Development**: set of modules $P = \{(M, T_M)\}$
- **Module dependency**: $M_1 \to^{\mathrm{M}} M_2$
- **Theorem dependency**: $t_1 \to^{\mathrm{T}} t_2$
- **Statement features**: $\mathrm{fea}(t)$

Dependencies are *direct* (one-step); $\leq_{\mathrm{M}}$ is the reflexive, transitive closure of $\to^{\mathrm{M}}$.

A development must obey the “well-formed imports” condition:

$$t_1 \to^{\mathrm{T}} t_2 \implies \mathrm{mn}(t_1) \leq_{\mathrm{M}} \mathrm{mn}(t_2).$$

Outline

Motivations

From Software Engineering to Proof Engineering

Proof developments, abstractly

Metrics for formal proof

Experiment and analysis

Conclusions

Out analogues of C&K

1. **WTM**: Weighted Theorems Per Module
2. **DIT**: Depth in Tree
3. **NOC**: Number of Children
4. **TDM**: Total Dependencies for Module
5. **CBM**: Coupling between Modules
6. **LCOM**: Lack of Cohesion in Module

Each metric is a function on proof modules.

TDM: Total Dependencies for Module

$$\mathrm{TDM}(M) = |\{ t' \mid t \rightarrow^{\mathsf{T}} t' \wedge t \in T_M \}|$$

TDM counts the number of theorems depended on in a given theorem's definition, both inside and outside the module.

Intuitively we expect that higher values may indicate more "brittle" modules.

CBM: Coupling between Modules

$$\mathrm{CBM}(M) = |\{M' \mid M \rightarrow^{\mathrm{M}} M' \vee M' \rightarrow^{\mathrm{M}} M\}|$$

High CBM indicates a module that is more closely bound in the hierarchy, suggesting it may be difficult to understand in isolation, or to move around.

LCOM: Lack of Cohesion in Module

Jaccard index measures statement similarity:

$$\mathrm{sim}(M) = \sum_{i=1}^{n} \sum_{j=i+1}^{n} \frac{|\,\mathrm{fea}(t_i) \cap \mathrm{fea}(t_j)\,|}{|\,\mathrm{fea}(t_i) \cup \mathrm{fea}(t_j)\,|}$$

LCOM is the average dissimilarity:

$$\mathrm{LCOM}(M) = 1 - \frac{\mathrm{sim}(M)}{\frac{1}{2}(n^2 - n)}.$$

High LCOM suggests a module that gathers together many unrelated things.

Outline

Motivations

From Software Engineering to Proof Engineering

Proof developments, abstractly

Metrics for formal proof

Experiment and analysis

Conclusions

Experimental study in three systems

The **Kepler formal proof**, FlySpeck in HOL Light, including HOL Light's libraries.

Isabelle HOL Main (core library), and three proofs: Auth, Bali, Probability.

The **Mizar Mathematical Library**, in particular the libraries of formalized topology and theory of lattices.

This experiment is non trivial! It needs specially modified versions of ITP kernels to gather proof objects, extract features and count dependencies. We reused a number of powerful tools from other work.

Average of metrics over large developments

	mods	thms	WTM	TDM	NOC	DIT	CBM	LCOM
Mizar MPTP2078	33	3646	110	270	9	10	26	74
HOL Light Core	21	2618	125	391	7	7	14	75
Multiv	19	11093	584	3091	7	8	34	76
Flyspeck	237	12999	55	1582	12	28	44	62
Isabelle Main	73	12731	174	357	8	18	18	72
Auth	38	4282	209	202	2	4	13	57
Bali	25	6946	502	261	4	5	17	67
Multiv	50	7821	287	245	2	4	17	61
Probab’y	45	5928	246	460	4	8	27	63

Average of metrics over large developments

	mods	thms	WTM	TDM	NOC	DIT	CBM	LCOM
Mizar MPTP2078	33	3646	110	270	9	10	26	74
HOL Light Core	21	2618	125	391	7	7	14	75
Multiv	19	11093	584	3091	7	8	34	76
Flyspeck	237	12999	55	1582	12	28	44	62
Isabelle Main	73	12731	174	357	8	18	18	72
Auth	38	4282	209	202	2	4	13	57
Bali	25	6946	502	261	4	5	17	67
Multiv	50	7821	287	245	2	4	17	61
Probab'y	45	5928	246	460	4	8	27	63

- Flyspeck: smaller modules, but not “shallow”

Average of metrics over large developments

	mods	thms	WTM	TDM	NOC	DIT	CBM	LCOM
Mizar MPTP2078	33	3646	110	270	9	10	26	74
HOL Light Core	21	2618	125	391	7	7	14	75
Multiv	19	11093	584	3091	7	8	34	76
Flyspeck	237	12999	55	1582	12	28	44	62
Isabelle Main	73	12731	174	357	8	18	18	72
Auth	38	4282	209	202	2	4	13	57
Bali	25	6946	502	261	4	5	17	67
Multiv	50	7821	287	245	2	4	17	61
Probab'y	45	5928	246	460	4	8	27	63

- Flyspeck: good cohesion (LCOM) but highest coupling (CBM).

Average of metrics over large developments

	mods	thms	WTM	TDM	NOC	DIT	CBM	LCOM
Mizar MPTP2078	33	3646	110	270	9	10	26	74
HOL Light Core	21	2618	125	391	7	7	14	75
Multiv	19	11093	584	3091	7	8	34	76
Flyspeck	237	12999	55	1582	12	28	44	62
Isabelle Main	73	12731	174	357	8	18	18	72
Auth	38	4282	209	202	2	4	13	57
Bali	25	6946	502	261	4	5	17	67
Multiv	50	7821	287	245	2	4	17	61
Probab'y	45	5928	246	460	4	8	27	63

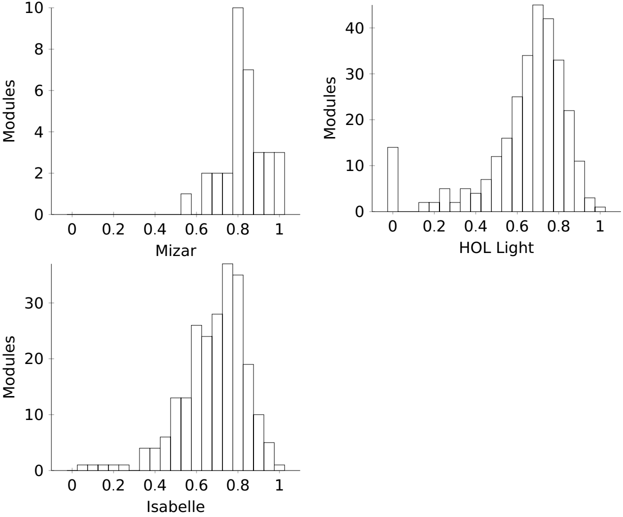
- Isabelle: WTM > than (thms/mods): more derived and auxiliary thms.

Average of metrics over large developments

	mods	thms	WTM	TDM	NOC	DIT	CBM	LCOM
Mizar MPTP2078	33	3646	110	270	9	10	26	74
HOL Light Core	21	2618	125	391	7	7	14	75
Multiv	19	11093	584	3091	7	8	34	76
Flyspeck	237	12999	55	1582	12	28	44	62
Isabelle Main	73	12731	174	357	8	18	18	72
Auth	38	4282	209	202	2	4	13	57
Bali	25	6946	502	261	4	5	17	67
Multiv	50	7821	287	245	2	4	17	61
Probab'y	45	5928	246	460	4	8	27	63

- Multivariate in HOL, Isabelle: look very different, why?

LCOM Histograms per system



Metrics over time

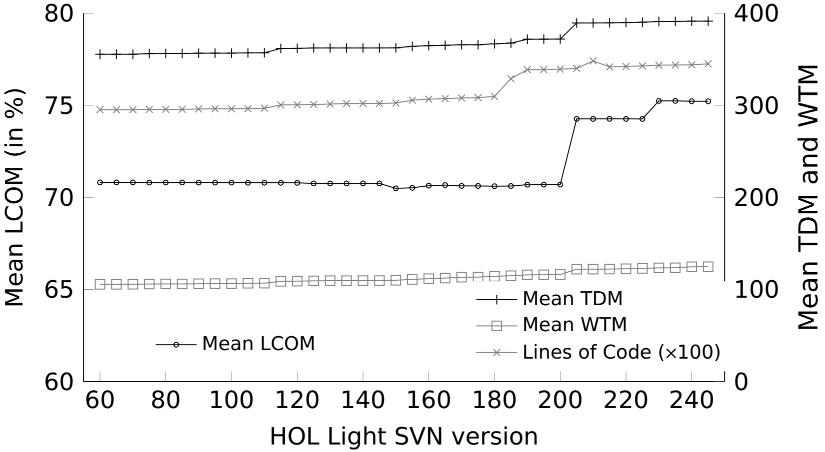
We examined five years of development of the HOL Light core library from its svn history.

Generally: see increase in sizes and complexity, sometimes improvement in cohesion.

Very stable code base so not the best test case. But we uncovered a change in modular structure, when a module `ind_defs` was removed.

We also examined the impact of a refactoring mentioned in the svn logs (see paper).

Metrics over time



Outline

Motivations

From Software Engineering to Proof Engineering

Proof developments, abstractly

Metrics for formal proof

Experiment and analysis

Conclusions

Conclusion

Proposed some first ideas for Proof Metrics.

- Inspired by well-investigated C&K Software Metrics
- Given formal definition
- Have some of Weyuker's properties (see paper)

Studied metrics over several large developments:

- Results restricted (core libs) but interesting
- Not argument for validity

A fairly new direction (see paper for related work).

Much more to do, even with current metrics/data.