"Mouldable Code" for Correct-by-Construction SW Needs Nested Theories — ≈6 years running Agda at the limits of the machine…

WOLFRAM KAHL

McMaster University, Hamilton, Ontario, Canada

17 September 2015

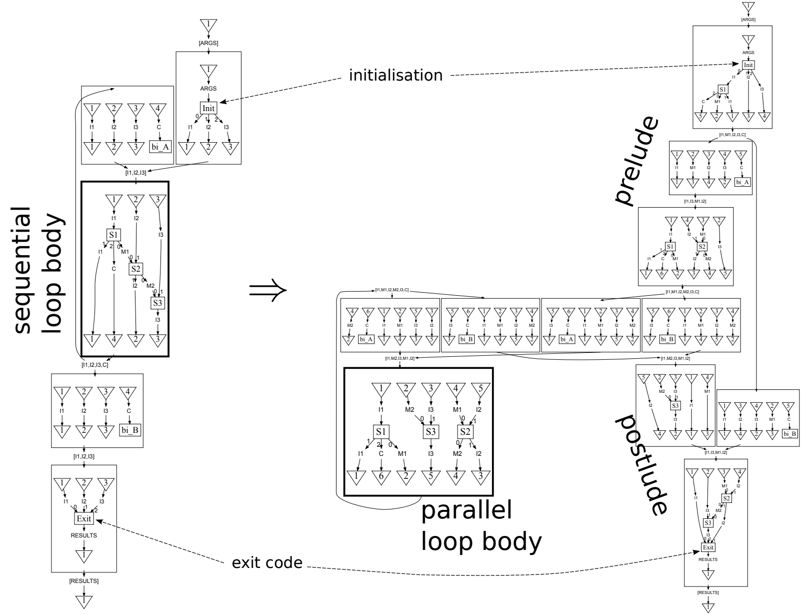
AIM XXII — Leuven, Belgium

This research is supported by the National Science and Engineering Research Council of Canada, NSERC.

“Mouldable Code”? — Background

- 1970s and 1980s at TU Munich: F.L. Bauer’s group: CIP
  - “Computer-aided Intuition-guided Programming”
  - CIP-L: Wide-spectrum language (functional — imperative)
  - CIP-S: (second-order) transformation system
- Gunther Schmidt’s reaction: **Transform Graphs!**
  - Term graph transformation system **HOPS** (several versions)
  - My PhD: second-order term graph transformation (used relation-algebraic formalisation and proofs)
- My Habilitation: Relation-Algebraic Approach to Graph Transformation
  - relation-algebraically amalgamated DPO and DPB
  - can handle “DPO + graph variables”
- **Coconut** (w. Christopher K. Anand): Software pipelining implemented as code graph transformation
  - generated “vector MASS” library shipped in IBM’s Cell BE SDK
  - implemented in Haskell
  - insufficient support by Haskell type system (no dependent types)

Software Pipelining as Nested Code Graph Transformation



initialisation

sequential loop body

⇒

prelude

parallel loop body

postlude

exit code

Transformation for Optimisation

- Many transformation patterns
  - are usefully **explained** as graph transformations
  - are normally implemented as AST transformations
- **Implementation** as graph transformations requires:
  - internal representation as graphs (not ASTs)
  - correctness of transformation wrt. graph semantics
  - sufficiently intuitive graph transformation concept

“Mouldable Code” [Gunther Schmidt, 1990s]

- Programs conceptually structured as graphs
- Program development is supported by a graph-based GUI
- Programs are written in a programming language that facilitates correctness proofs
- Program development is supported by a powerful transformation system that allows power-users to “turn the programs inside out”
  - for the purpose of fusion and other efficiency-improving adaptations
  - and also for systematically and without impacting correctness adding what would later become known as “aspects”
- The resulting programs are **correct by construction**

Nested Code Graph Transformation

- Control-flow graphs: Kleene algebra
  Kleene categories
- Data-flow graphs: gs-monoidal categories
  (tabular allegories)
- **Equations** turn into **transformation rules**
- Matching implemented as graph homomorphisms
- Transformation via variant of DPO approach
  - Correctness wrt. gs-monoidal categories: Zhao Yuhang
  - Correctness wrt. Kleene categories: TBD
- **One-directional rules** can be used for **refinement**
  (demonic) Kleene categories

Getting Started — Essential Ingredients

- **RATH-Agda (≈500 pages):** Abstract formalisation of semigroupoids, categories, allegories, Kleene categories, collagories, action lattice categories
  - Relatively fine-grained hierarchy of theories
  - Many module splits for performance reasons
  - Allegory and category combinators still slow (>9GB heap)
- **SUList (≈200 pages:)** Sorted unique lists
  - Directly implement sets
  - Key-value-pairs: Finite maps
  - Set-valued maps: Finite relations
  - Invariant-carrying datatype, no irrelevance
  - Many correctness proofs involve large case analyses
  - ≈4GB heap
  - ListSetMap implements Kleene collagory; sub-category of mappings equivalent to FinVecCat
    — ≈10GB heap
- **JSON Parsing and Pretty-printing (≈100 pages)**

It Calculates a Pushout! — in 6 seconds...

- A single top-level module brings the three strands together
- Can read and write graphs in JSON format
- Calculates a small (**6 node**) pushout
- MAlonzo:
  - Compilation to Haskell (after typechecking): 40min, >4GB heap
  - GHC call: 40min, >7GB heap
  - Binary size 160MB ; run-time: **≈6s**
  - Probable problem: No compromises:
    Invariant-carrying datatypes, no **abstract**, no irrelevance
- UHC (March): Binary size 60MB; segfaults
- UHC whole-program optimisation (-O2,2,2, March):
  Binary size 7MB; run-time: >**5min**

Term Graph Decomposition

- Yuhang Zhao implements term graph decomposition into gs-monoidal category expressions [Corradini, Gadducci 1998]
- Concrete model: 2-Category of Term Graphs on top of FinVecCat:
  - Correctness proof involves three levels of categories: Holes unusable
- This is an essential ingredient to proving correctness of DPO term graph rewriting wrt. functorial semantics

Side-Show: AContext

- Abstract formalisation of FCA context categories only needs OCC with powers, residuals, and symmetric quotients
- Agda used as **“just a mechanised mathematical notation”**

  that lets me write the mathematics in a natural way
- The **abstract algebraic style** plays to the strengths of Agda
- 189 pages
- Final chapter: Finishing off categoric duality between FCA contexts and complete lower semilattices:
  - Duality proof runs out of 52GB heap
  - One-line definition of the back-and-forth functors takes hours to type-check
    - Issue 1625
    - Andrea Vezzosi supplied experimental patch
    - Will try this week: Does this also help me elsewhere?

Ceterum Censeo . . .

. . . *cum grano salis* . . .

- Agda got many important things right, and has been improving tremendously
  - but even from Agda-dev, I don't get a feeling where Agda is headed
- We need a roadmap towards a trusted kernel
- We need an "Agda report", perhaps initially limited to the trusted kernel
- We need a roadmap towards self-hosting — Agda in Agda
  - AIM as "Agda hackathon" would profit from the confidence of producing Agda code!
- We need efficient compiled code
  - We need whole-program optimisation
  - We may need *semantics-preserving* pragmas to guide optimisation — not extensions like irrelevance

Ceterum Censeo . . . (ctd.)

- First-order sharing is probably not sufficient for efficient type-checking of level-polymorphic code?
- Agda's module system is wonderful to use!
  - Am I the only one using it in certain ways?
  - Documentation of performance implications is needed
  - Nested parameterised modules probably still have problems (Issue 1396)
    — who else besides Ulf understands the implementation of the module system?
    What would it take for me to understand it?
- Sometimes I look at Agda implementation modules, and lack (pointers to) documentation. . .
- **I ♡ Agda**