ISSN 2186-7437

NII Shonan Meeting Report

No. 2011-2

Agda Implementors Meeting

Yoshiki Kinoshita
Peter Dybjer
Shin Cheng Mu

September 8–14, 2011

National Institute of Informatics
2-1-2 Hitotsubashi, Chiyoda-Ku, Tokyo, Japan

Agda Implementors Meeting

Organizers:
Yoshiki Kinoshita (AIST)
Peter Dybjer (Chalmers Unviersity of Technology)
Shin Cheng Mu (Academia Chinica)

September 8–14, 2011

14th Agda Implementors Meeting (AIM XIV) took place as one of NII Shonan Meeting. By tradition, AIM had been held for a period of 7 days, so some participants arrived a day earlier and started the meeting as "non NII part" of the AIM XIV. Moreover, the last day, 14 September, was dedicated as the "DTP-AIM joint day," where DTP (Dependently Typed Programming) was another NII Shonan Meeting held from that day. The joint day was planned because the two meetings has close direction of interest and there were considerable number of persons participating the both meetings.

AIM consists of code sprints and talks. Code sprint is recorded in Agda Wiki:

`//http://wiki.portal.chalmers.se/agda/pmwiki.php`

but it is printed in the following for readers without easy access to internet.

Overview of Code Sprints

On the first day, there was two hour discussion, attended by all participants, about what to do in code sprints. Every participant is asked, one by one, to declare one or more themes which he/she wishes to do during AIM. Then, on the next round, participants are asked to declare whether he/she would like to join other themes declared. Project teams are formed that way, and the leader for each team is assigned. This is what is done in the morning on the first day.

Then, during code sprint time, every participant participates one or more project. At the end of each day, we had a wrap-up hour, and the leaders report what was done on the day. On the last day, the whole afternoon was dedicated to wrap-up. The note taken during the wrap-up is recorded in Agda Wiki, which is listed below.

```
* Irrelevance
  More Polymorphism,
  Consistensy,
  Irrelevance at universe level.
  Problems in inferring irr. / termination checking.
** Mem: Andreas*, Gyesik, Bengt, Yoshiki, ...
** Day1:
   Not started.
** Day2:
   Andreas: gave a Talk.  Looking at Consistency issues.

     -- bug because Termination Checker doesn't look inside rec call.
     rel : .\bot -> \bot
     rel ()

     false : \bot
     false = rel false

   Andreas gave a demo for a simple example of irrelevance.
** Day3:
   AA: Bug with Record with only irr. fields fixed.
   More features requested at IRC.

   The bug with termination check with rec call fixed.
** Day4:
   In all, 11 bugs are fixed and committed (cf. bug track)

   AA:
   Hetrogenous equality can correctly compare elms of isomorphic types.

      data B : Bool set where
        tt : B true
        ff : B false

      succeed : tt =Heq= ff -> \bot
      succeed ()

      data Unit1 : set where unit1 : Unit1
      data Unit2 : set where unit2 : Unit2

      fail : unit1 =Heq= unit2 -> \bot
      fail () -- fail. (and should fail)

   (Ulf: Rather, the problem is to disprove Unit1 =Heq= Unit2.)

   Records with all fields irr.

    record Setoid c l : Set (suc (c lub l)) where
      field
        Carrier : Set c
        _~_ : Rel Carrier l
        .isEquivalence : IsEquivalence _~_   -- if you make this irr,

      open IsEquivalence isEquivalence public -- this used to be
illegal.

   Now,
    record R1 : Set1 where
      field .f1 : Set

    record R2 : Set2 where
      field .f2 : R1
            f3  : Set
      open R1 f2 public  -- is ok only when f1 is irr.

   (Ulf: IRC ppl still not happy. They want to open R1 making f1 irr. )

   More generally, a question of distributing irrelevance to record
   fields.  The case of non-dependent record is thinkable.  Dependent
   record case is questionable.  A field value can be irrelevant
   but the type of later fields containing it can't be.

* Formalizing semantics of Regular Expression Matching
  ... had done the same in Isabell; equiv. of sem and impl.
** Mem: Minamide*, James, Nisse
** Day1:
   Stil learning Agda.  List Monad.
   Struggling with manual rewriting that was automatic in Isabelle.
   Ulf: Lib for eq rewrite, and,
        "rewrite" construct should help.
        eq : t \equiv u

        f : (x : P t)... -> G t
        f x y z rewrite eq = ... rhs[here, type of x is P u, goal is G
u] ...
        See doc/release-notes/2-2-6.txt
** Day2:
   Minamide: Interesting discussion with Nisse.
   Compared Minamide-semantics and Danielsson-semantics.
   Formalised sem in Agda.
   Proved it's terminating using accessiblity predicate.

   data RegEx ... -- usual

   mutual
     match : (N -> N -> Bool) ->
             (r : Regex N) ->
             (w : List N) ->
             Acc Lex._<_ (r , w) ->
             List(Sigma (List N) (\ w' -> w' <<= w))
                                              -- isSuffixOf

   (Finished code will be uploaded somewhere?)

* Assurance Case in Agda
** Mem: Yuasa*, Bengt, Makoto(AS), Yoshiki
** Day1:
   Yuasa: installing D-Case/Agda
   B,M,Y: Discussed the possibility of using "exceptions"
            to express the possibility of hidden assumptions
            that are normally presumed holding.

            Instead of A -> H -> B,  think A -> (B or not H)
** Day2:
   Yuasa: Translating AC of an internet server into D-Case/Agda.
          (Showing a graph version and Agda version.)
** Day3:
   Yuasa: Almost finished translating the structure of the original Case
to D-Case/Agda.  Formalising the contents tomorrow.
** Day4:
   Yuasa: Finished translating the original Case to D-Case/Agda.
   (the code and the graph shown.)
   Editing the EMF xml code into agda term semi-mechanically.

   Instead of formalising the contents of goal nodes etc(too
   complicated for now), going for an "abstract framework" to express
   system change(?).

   Using co-algebra to express the life-cycle of the target system.

     -- s < s' = s is improved to s'
     data OSD : {Sys : Set}(_<_ : Bin Sys) : Sys -> Set where
       osd : (s s' : Sys)-> s < s' -> \inf(OSD _<_ s') -> OSD _<_ s

     ca-process : ... ->
         (s0 : Sys) -> ((s : Sys) -> \Ex (_<_ s)) -> OSD _<_ s0


* Inference for Recursive Types, Unification
** Mem: Jacques*, Nisse, Andreas
** Day1:
   JG: Translating conor's unification algorithm in Agda.
   (Nisse: new work in XXX possibly similar.)
** Day2:
   JG: Going on w/ unif.  Fixing the "yellow boxes".
   Overlapping pattern can be a nightmare.

   (Nisse:)

      true  =?= true  = true
      false =?= false = true
      _     =?= _     = false

      -- now try to prove property of =?=
      prf true   true = ...
      prf false false = ...
      prf _     _     = ...(here, type checker knows nothing about
(_=?=_), not even reducing to false.)...
```

Cf. Coq's desugaring to real branches. (-> quadratic blowup.)
** Extra Day: It's not so easy to separate the structure from the
Proof. Some thinking occurred in Jacques' head.
** Day3:
Going on with unification. The code is on the bug tracker.

UN: Since 2-2-10, examples are a lot faster, but this particular
example was three times slower -> bug tracker.
** Day4:
Can prove soundness of unification.
Clever encoding of substitution using deBrujin ix.
Following Conor's paper.
Begining was easy ...
Some lemma easier to write than to find in Library.
Equality reasoning too verbose to write.

Example of code which is three times slower with Agda head ver.

```
     -- accumulating mgu spec...
     amgu-P s t u u' = unifies s t u' X u' sub u X (\all v -> v sub u ->
unifies ...)
     amgu : ...
     mgu : ...

     amgu-ind : ...
```

Wish for Agda development as Coq user:
- the strongly dependently typed way of writing function needs
getting used to.
- Big case analysis more cumbersome than Coq.
- in the end, not much difference with Coq.
- Obstacles: knowing the language itself - the ref.man. is weak.
  knowing how to use features like "with" and "rewrite". Hard to
grasp what abstractions are done behind the scene.
  (NAD: can explicity generate helper functions interactively.)

* Normalization
** Big step NBE
NBE like, but with a general recursive algo & termination proof for
* it.
*** Mem: Gysiek*, James
*** Day1:
GL: Reading the paper by JC.
(JC: superceded by the Thesis.)
*** Day2:
JC: Updating Agda formalization from the older code.
Will put on Wiki later.

```
*** Extra Day:
   Finished for now. Uploaded to github
* [[https://github.com/jmchapman/Big-step-Normalisation]].
*** (Day3: conclusion)
   Dissapointing performance: not much memory improvement over 2.2.6?
   UN&NAD: possibly universe polymorphsim?

** Yoneda NBE
   Relationship between structure and necessary properties of equality
*** Mem: Yoshiki, James,
*** Day1:
    YK: Dug up the old code for natural transformations.
*** Day2:
    YK sent paper to JC, not read yet.
    Some discussion with MT on Setoid-enriched categories?
    JC: Will show categories without setoids.
*** Day3:
    YK: Code written up to Natural Transformation.
    (Code shown.)

      record Cat (a b : Level) : Set(suc(a u b)) where
        field
          object : Set a
          hom : object -> object -> Setoid b
          comp ...
          id ...
        ...

    (NAD: IRC ppl are developing cat thy with irrelevance.
    see https://github.com/agda/categories    )

    (NAD: Formalising Cat Thy is a right of passage for students of
     Agda (and Coq and ...))

    Notations: ...

    (NAD: Why work with Setoid rather than just postulating extensional
quotient -- JC: HOW?.)

    Re: Universe level lifting.
    UN: We could get rid of the constructor lifted : A -> Lifted A,
    but we won't be able to get rid of Lifted itself.
*** Day4:
    (YK showing the final code.)

      \Phi : {a b : level}
             {C : Cat a b}
             {x y : ob C}
             [C op , SETOID a b ] < C <-, x > , C <-, y>> -> C < x , y >
      \Phi t = ...
      \Psi : ... (inverse dir) ...
      \Psi f = ...

      ... and the proof that \Phi and \Psi are inverse to each other
          with respect to the equality of respective hom setoids. ...

    The code will be uploaded after remaining metas for eq prfs are
solved.

    IRC ppl also formalising Cat Thy.


** NBE for?/without? Large Elimination
*** Mem: Nissse, James,
*** Day1:
    N: Looking at the code.
*** Day2:
    NAD&AA Discussion. How AA proves termination of his algo.
*** Day3:
    NAD:  Postponed.

* Performance Drive
** Doc wiki page
   Hints for avoiding performance limitations, checking a large set of
* files, etc.
*** Mem: James*
*** Day1:
    JC: Info gathered on the white board.
*** Day2:
    JC: DONE! Page [[Main.PerformanceTips|Performance tips]]
    (Going through points)
*** extra day:
    JC: updated. Linked to the wiki under "Documentation-> Howto".

    Discussion on "abstract"
    Ulf: (explaining why it makes typechecking quicker,
          why "first try without unfolding" is too brittle.)

    JC: Old fashioned:
         {-
         prf : A
         prf = ...(real)...
         -}
         postulate prf : A

    rel between irr and abstract.

** Prototype to experiment with variables and sharing
*** Mem: Nisse, Ulf*, Andreas
*** Day1:
    Talking
*** Day2:
    Hacking
*** Day3:
    Nothing
*** Day4:
    Delayed

** New constraint solver
*** Extra Day: New infrastructure implemented (but does not improve
** performance yet). More infrastructure needed.
*** Day3:
    (New part: constr. managmement. wasteful e.g. waking up useless
** constr.   Now try to wake up only if there is a chance of making
** progress.  In the middle of thinking of how, we came upon nicer
** infrastructure.  Once got up to the point something can be type
** checked, but that was wrong(?) -> back to coding now.)
*** Day4:
    UN: All the infra is in place.  Some bugs remain.  Will be fixed
** soon.

    The new infra machinery:

    This happens all the time (*).

       t : B  A = B
      --------------
         t : A

    Sometimes, the check A = B must be postponed and kept as a
constraint.

    Old way:
    - everytime any meta is instantiated, all the constr. are waken up
and re-checked.
    - all the constr. are passed around in various places.

   New infra:
   - sleeping constr. and awake constr. separated.

      f ?a = t in sleeping

      [?a := suc n   instantiated]
      [sleeping constrs. scanned for ?a]

      f (suc n) = t moved to awake contr.

      Within the sleeping, no ordering for now.  Shouldn't matter
      for correctness, may matter of performance.
   - The check A = B just possibly add to the sleeping list
     instead of possibly returning the constr to be passed around.
     Much nicer when comparing wrt telescopes.

       a ; as  =?= a' ; as' :   A ; AS

     If
       a = a' : A
     is postponed, old way produced a guarded constraint
       { as =?= as' }[ a = a' : A]
     This compound constr. does not fit into the new infra.
     In the new way, each constr belongs to a "problem".
     For a guarded contr., instead of guarding constraints themselves,
     "problem id" tag is put.  A tagged constraint can be tried to be
     solved only when the constraint stack does not contain
     the constr. that belong to the tagged problem.

   (some explanation of the constrs you see in the constr buffer for
internally generated metas)
   When the check A = B in (*) generates constraints,
   the check t : A returns the fresh meta ?t : A, with constr
   ?t = t that's blocked on A = B.
   The new infra would make this blocking relation more readable.

   (some possible explanation on the tc slowness of JG's code:
    metas at the top of the file may be generating lots of constrs.
    ->
    New infra should be more efficient.  Besides, Nicolas' prohibition
    on going past yellow should be considered.)

* Support for more liberal inductive recursive types
** Mem: James*, Andreas, Ulf
** Day1:
   not started.
** Day2:
   Talked about what's the state of affairs.
   Backward compatibility -> to be ditched.
** Day3:
   JC: Merged Andreas' patch from AIM13 with the latest head.  Not yet
* tested.
** Day4:
   JC: Kind of works now.

     mutual
       data U : Set    -- no where        -- these doesn't have to be
       El : U -> Set   -- no clauses      -- at the top.

       data U where  -- no type
         unit : U
```

```
          pi : ...

        El unit = Unit  -- no type sig.
        El (pi s t) = ....

    -- nice to show the judgment forms at the top.
      mutual
        data Con : Set
        data Ty : Con -> Set
        data Tm : forall \GG -> Ty \GG -> Set
        ...
        data Ty where -- no parameter decl. -- problem?

    -- implicit params must be listed in def of data and record.
      data X {Y : Set} (Z : Set) : Set
      data X {Y} Z where
        x : X Z

    NP: Want to interleave constors of different data's and fun clauses.
    NAD: Separating typesig and def is fine, but separating
    parts of a def seems weird.
    UN: problem impl wise, too.

    JC,UN,NP found a possible solution(?)
    NP:
       mutual
         -- just the typing first.
         Con : Set
         \eps : Con
         ...
         data Con where \eps   -- no typing here.


    Works for record types too.
      mutual
        record R1 : Set

        record R2 : Set where
          field ...

        record R1 where
          field ...

    Fixity info for record constructors is now thrown away; to be fixed.

    (A corner case when embedding a univ to another.)

  * Incremental Syntax Highlighting
    (More responsive emacs interface Checking and editing asynchronously.)
  ** Mem: Guilhem*, Nisse
```

```
** Extra Day: Incremental Progress (Arbitrarily small).
** Day 3:
   building a new communication method between Agda and emacs for
   hightlighting info.  On the .el file side now.
** Day4 :
   The Haskell part is nearly done.
   Demo with simulated highlight info generation.

* Matrix Multiplication
** Mem: ?* (James, Yoshki, Andreas)
** Day1:
   Demoed.  Files somewhere. Done

* Agda Intro Sprint
** Mem: Makoto*, Yoshiki, Bengt
** Day1:
   not started.
** Day2:
   Ibid.
** Day3:
   Only outline plan: Intro as a normal func lang; Agda
* characteristics(point and click, manual proof writing for human
* consumption); Agda History; Agda lang.; DTP; Application to
* Argumentation.

   Dependent Indexed Monad and typed exception?

      A -> E1 + B
      B -> E2 + C
    ---------------------
      A -> (E2 + E1) + C

      (x : A) -> E1 x + B x
      (x : A) -> (y : B x) -> E2 x y + C x y
    -------------------------------------------
      (x : A) -> ?

   NAD: Instead of input, have extra index like
     {i} A_i -> E1_i + B_i
** Day4:
   Nothing

* Windows Installer
** Mem: Makoto
** Day1: Done.

* Discussion on definition in record decl. / in let.
** Mem: ? UN, JC*,
** Day2:
   Ulf: explaining how record generates a special datatype and
```

```
        a module.
   Let-delcs could be general, but generative equality would be
   too hard to explain.
   -> Introduce structural equality?

* Epigram "by"?
** Mem: JC, Nisse
** Day3?:
   In Epigram, user chooses induction principle to use.
   Right now, can be done with library, but semantic sugar
   would be nice.

* Agda Manual
  To be finished by AIM 24
** Mem: PD
** Day3:
   AIM: "Agda for working logicians"
   For those people who are looking for the OLDEST features.

   Keeping in touch with the logical Martin-Lof core of Agda.
   Various addition (Ind.Def, Coind, etc) coming from various places.
   Intro these gradually.

   Each type former introduces a new feature, e.g. empty type -> pattern
"()"

   Feedback welcome.
** Day4:
   Wrote 10 page draft. "Martin-Lof Type Theory in Agda"
   Along the line of the NPS book.

   Using the oldest (most basic) feature to gradually building up.

   ... grammar of terms ... [Aside: \ x -> or \x -> ? ]
                            [Aside2: Add footnote about Agda space
peculiarity]
   ... The Theory of Sets ...
   Empty set N0 w/ R0       [UN: (x:A)(y:B)-> is better than
(x:A)->(y:B)-> ]
   ...N...W... whole MLTT [Upto here done]...
   ... Extension to lambda ...  Records ... Inductive definitions ...

   The source of the draft is an .lagda file, but not type checkable
   as it is.

   What's next?

   (Discussion of presenting agda terms while not mentioning meta
etc...)
```

```
   The draft will be placed in XXX

* Turn a yellow thing to an error if Agda can tell it is an error.
** Mem: NP
** Day3:
   Arrived.

* Formalizing CPS Transformation
** Mem: KF
** Day3:
   Arrived.

* Formalising the rel btw Indexed sets (Fam) and Fibration (Pow)
** Mem: Makoto Hamana
** Day 4:
     (p : E -> I) <---> (p\inv : I -> Set)
   The bijection was proved using JC’s library with proof irrelevance
* and extensionality.

* Talk: 3 slots in AIM, 2 in Joint day.
  Andreas: irrelevance. - Fri.
  James: Relative monads - Mon
  XXX: - Tue

  Nisse: NBE in Joint Day.

* (Live coding session:
  Day1: Andreas:-> Nisse)

* Release Engineering for Agda 2.2.12
  Bug fixes etc.
** Extra Day: many issues closed!
** Day4:
   During AIM14, 11 bugs are fixed.
   Now some cooling down period before the release of 2.2.12.

   Remaining: 5 high prio defects and lots of enhancements reqs.

* Discussion on Next Agda Meeting:
** Day3: The Situation.
*** The current plan: 1hr from Munich, Hotel AURACHHOF, Skiing AIM
   19 Feb (Sun)-> 26 Feb   (changed from unavailable 9 Feb -> 16 Feb)
   85~90 euro/person*day
   Hotel needs confirmation by the end of this week.
   Expecting about 30 ppl.
*** !! Crashes with IFIP 2.8 where Ulf is the invited speaker and having
* baby. !!
   BN: Skiing less important than having Ulf.
*** => Move to Apr?
    Will talk on Day4 again.
```

```
** Day4: The Decision.
   If Apr, must be before 16th(AA teaching), which is sort of Easter.
   After that, Summer.

* Discussion on possible name change to attract more ppl.
UN: the character of AIM more important than the number...
BN: Good to have a place where USERS can meet and share experiences.

(the problem of money for them to attend)
(there are IRC etc. users are already gathering)
```

Overview of Talks

Following talks were given during AIM XIV. In average, one talk is given per day, in the morning. The last four talks (Nordström, Hamana, Danielsson, Sato) were given on the "DTP-AIM joint day."

On Shape Irrelevance and Polymorphisms in Type Theory

Andreas Abel, University of Munich

We introduce the concept of shape irrelevance in the context of dependent type theory. A type valued function is shape irrelevant in its argument if the argument does not influence the shape of the type. For example parameter and index arguments to the inductive family of vectors are shape irrelevant since the result is always a vector type (although of different length or element type). In contrast, large eliminations are not shape irrelevant. The concept of shape irrelevance allows to declare type arguments as computationally irrelevant, thus, reducing the amount of conversion checks during type checking and eliminating static parts of programs during code generation. In the talk, we present a type system for shape irrelevance and discuss its integration into Agda.

Relative Monad

James Chapman, Institute of Cybernetics, Tallinn

Towards a computational interpretation of parametricity

Jean-Philippe Bernardy and Guilhem Moulin, Chalmers University of Technology

Reynolds' abstraction theorem has recently been extended to lambda-calculi with dependent types. In this paper, we show how this theorem can be internalized. More precisely, we describe an extension of the Calculus of Constructions with a special parametricity rule (with computational content), and prove fundamental properties such as Church-Rosser's and strong normalization. The instances of the abstraction theorem can be both expressed and proved in the calculus itself.

Proof Theoretic Perspective on Dependently Typed Functional Programming

Bengt Nordström, Chalmers University of Technology

Dependent Polynomial Functors for Inductive Families

Makoto Hamana, Gunma University

It is well-known that every algebraic datatype is characterised as the initial algebra of a polynomial functor on sets. We extend the characterisation to inductive families. Specifically, we show that inductive families are characterised as initial algebras of Gambino and Hyland's dependent polynomial functors. We also show that the differentiation of dependent polynomial functors provides zipper data structures on inductive families. This is joint work with Marcelo Fiore.

Normalisation for "Outrageous but Meaningful Coincidences"

Nils Anders Danielsson, Chalmers University of Technology

Last year McBride presented a new technique for defining dependently typed languages in a well-typed way (without using raw terms). However, he left the definition of normalisation functions to future work. I will show how one can implement a normalisation function for a simple, dependently typed logical framework defined using McBride's technique.

Ez is Easy

Masahiko Sato, Kyoto University