Extracting Theory Graphs from Aldor Libraries (System&Data Paper)

Florian Rabe, Stephen Watt

September 2023

Our Results in One Slide

Setting

- Aldor = computer algebra system   arose from Scratchpad II, Axiom
  - several interesting type system features
  - significant library ($>$ 300 files)   unlicensed, closed source
  - small user community
- theory graphs = diagrams of theories and morphisms

  e.g., *Group* $\to$ *Ring* $\to$ *Field*
- MMT = logic-independent MKM language based on theory graphs

Result

- represented Aldor language in MMT
- exported Aldor library as MMT theory graph

  440 theories and morphisms, open source

Dual Contribution

Aldor export

- makes Aldor more well-known, available
- enables KM services for Aldor — e.g., search, library browser
- starting point to integrate Aldor with other systems/libraries

solid work, but unsurprising
kind of boring to present

Design type systems for math

- proof assistant type systems $\neq$ computer algebra type systems
  very different
- to be expected
  - CASs designed for (a fragment of) math
  - PAs mostly designed for software verification
- but embarrassing how badly we understand the difference

more interesting to talk about

Aldor Features: Theories as Types

Principal concepts

- categories: special types = theories, record types

```
define Group: Category == Monoid with {
   /: (%, %) -> %;
}
```

- domains: elements of categories = models, record elements

```
define IntegerAddition: Group == add {
   Rep == Z;
   *(x:%, y:%):% == ...;
   1: % == ...;
   /(x:%, y:%): % == ...;
}
```

- values: elements of domains `x:IntegerAddition` refers to an element of the representation type of `IntegerAddition`

dual role of domains: elements and types

Aldor Features: Parametric Theories, Functors

Toplevel definition = function

- typed arguments
  - domain variable (typed by some category)
  - object variable (typed by some domain)
- typed return value
  - new category = parametric theory, dependent type
  - domain of some category = theory morphism, functor

```
define ResidueClassRing(R: CommutativeRing, p: R): Category ==
  CommutativeRing with {...
    modularRep: R -> %;
    ...
  }

define IntegerMod(Z:IntegerCategory, p:Z): ResidueClassRing(Z, p) == add {
  Rep == Z;
  modularRep(r:Z):% == per(r mod p);
  ...
}
```

Aldor Features: Soft Records

Soft typing well-known

- type membership checked dynamically
- e.g., e:prime depends on run-time value of e — undecidable typing

```
define ResidueClassRing(R: CommutativeRing, p: R): Category ==
  CommutativeRing with {
    if R has SourceOfPrimes And (prime? $ R) p then
      Field;
  }
```

Now: presence of a declaration dynamic

- R has SourceOfPrimes soft-typing check on a category type
- (prime? $ R) p if so, prime? available on R
- if ... then Field and if p is prime, the resulting ring also inherits category Field

presence of declarations on ResidueClassRing(Z,x) undecidable

Aldor Features: Soft Records (2)

Presence of declarations depends on context

```
define Complex(R: ArithmeticType): ArithmeticType == add {
  complex: (R, R) -> %;
  ...
}

extend Complex(R: ArithmeticType):
  LinearArithmeticType R with {if R has Field then Field} ==
    add {
      ^(p:%, n:Integer):% == ...
      if R has Field then {
        inv(a:%):% == ...
    }
```

Add declarations to a domain/category

- extend interface when using it, e.g.,
  - to keep definitions in a different file
  - to add to another author's definition
- extend interface when arguments have sharper types

Representing Aldor in MMT

Aldor Language

- MMT theory *Aldor* declaring all Aldor primitives

```
theory Aldor
  type
  ...
```

- currently only Aldor syntax represented
- future work: Aldor type system, logic, computation

Categories

- special MMT theory

```
theory Category =
  include Aldor
  %:type
```

- categories that use % $\rightsquigarrow$ theories that include Category
- category-valued function $\rightsquigarrow$ parametric theories

Representing Domains in MMT

Basic domains

- domain D of category $C \rightsquigarrow$ theory morphism $D : C \to \mathit{Aldor}$

```
define PointedSet == add {c: %}
define NatZero: PointedSet == add {Rep == Nat; c: % = 0}
```

$\rightsquigarrow$

```
theory PointedSet =
  include Category
  c: %

morphism NatZero : PointedSet -> Aldor =
  % = Nat
  c = 0
```

- Aldor representation type $\rightsquigarrow$ definition of special constant %

Representing Domains in MMT (2)

Domain-valued function D

- invent special theory for the arguments

```
define D(ARGS): C == DEFS
```

$\rightsquigarrow$

```
theory D_args =
  include Aldor
  ARGS

morphism D: C -> D_args =
  DEFS
```

MMT limitation: awkward name generation required

Representing Soft Records in MMT

Conditional declarations

- invent nested theory with condition as axiom

```
define category C == {... if p then DECLS ...}

⇝

theory C =
  ...
  theory C_cond_1(condition: p) =
    DECLS
  ...
```

awkward non-canonical name generation C_cond_1

- given domain d of C satisfying p
  access of DECLS ⇝
  composition of d:C −>Aldor with
  retraction from pushout(d,C_cond_1)(proof of d(p)) to C

Conclusion

Overview

- exported 321 Aldor source files as 440 MMT theories/morphisms
- a few advanced Aldor features
  - unsupported
  - eliminated in intermediate representation provided by Aldor
- key insight: Aldor type system very nice but
  - deserves modern reinterpretation
  - not directly representable in modern systems

MMT representation helps with both

Future Work

- expand MMT to support Aldor-like features more naturally
  - theory/morphism-valued functions
  - smoother handling of conditional declarations/extensions
- represent Aldor semantics via logical framework
- use Aldor export for interoperability, KM applications