Type Theory in a Membership Equational Logic Framework

Mark-Oliver Stehr

SRI International

Motivation

What is the most simple and natural framework

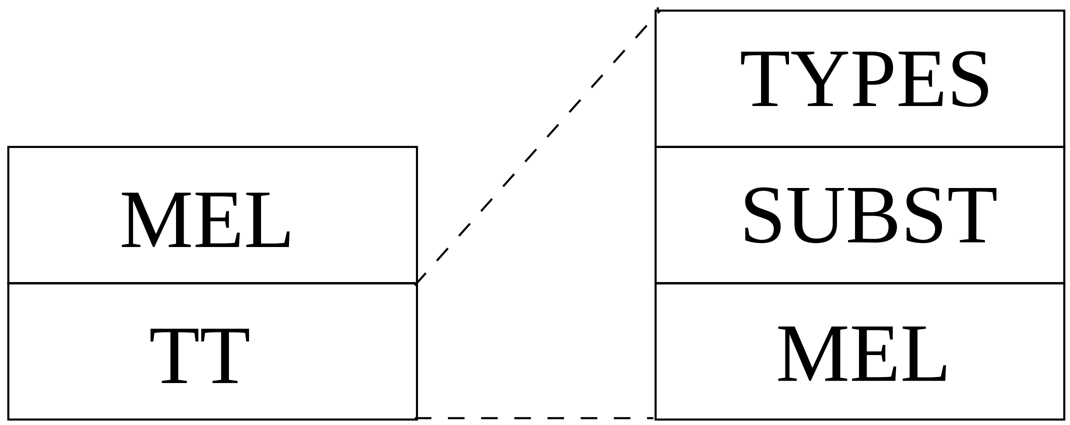
- *for* a rigorous (operational) explanation of higher-order type theory (object language)
- *without* presupposing complex notions of type and binding ?

Our Answer: **Membership Equational Logic (MEL)**

Here MEL is a *meta-language* for the type theory !

Membership Equational Logic (MEL) is a minimal algebraic type theory.
Can it be naturally extended by higher-order features and complex types ?

In this case MEL becomes a *sublanguage* of the type theory !



More Motivation

- Rewriting, the computational mechanism of MEL,
  is at the core of virtually all type theories and theorem provers.
- MEL provides an algebraic notion of mutually inductive definitions with equations.
- But: Conventional rewriting is too restricted to compute with non-free algebraic datatypes. We need rewriting modulo equations, like associativity and commutativity.
- Rewriting Logic (RL), the generalization of MEL, provides a fundamental link between computation in functional programs and computation in concurrent, nondeterminstic systems.
- RL provides a link to other paradigms such as
  concurrent object oriented programming and high-level Petri nets.
- The Maude rewriting engine implements MEL and RL in a highly efficient way.
- Maude provides additionally overloading, a mixfix parser, and support for reflection.
- Rewriting has good potential for parallelization.

Experiment: Calculus of Constructions

Binding and Substitution

- We generalize the indexed-based explicit substitution calculus of Pierre Lescanne.
- The resulting calculus unifies names and indices.
- The de'Bruijn representation becomes the so-called α-normal form for a suitable reduction system.

Type Theory

- Starting point is the Generalized Calculus of Constructions
  (with two impredicative universes and an infinite cumulative hierarchy of predicative universes)
- We have extended the calculus by type annotations, more general contexts and definitions.
- The executable MEL specification gives a type checker which
  is also a higher-order logic proof checker according to the propositions-as-types interpretation.

Excerpts from the Specification I

```
subsort Qid < Var .
subsort Var < Trm .
subsort Con < Trm .
subsort Def < ContextEl .
subsort Abs < ContextEl .
subsort Subst < ContextEl .
subsort ContextEl < Context .
```

```
op _{_} : Qid Nat -> Var .
op [_=_] : Qid Trm -> Def .
op [_:_] : Qid Trm -> Abs .
op {_:_} : Qid Trm -> Abs .
op __ : Trm Trm -> Trm [gather (E e) ] .
op __ : Abs Trm -> Trm [gather (e E) ] .
op __ : Def Trm -> Trm [gather (e E) ] .
op __ : Subst Trm -> Trm [gather (e E) ] .
op _:_ : Trm Trm -> Trm [prec 45] .
op [_:=_] : Qid Trm -> Subst .
op [lift__] : Qid Subst -> Subst .
op [shift_] : Qid -> Subst .
op ? : -> Qid .
op ERRTRM : -> Con .
```

Excerpts from the Specification II

```
op substreduce : Trm -> Trm .
op compatible : Context Trm Trm -> Bool .
op type : Context Trm -> Trm .
...
eq type(C,(M : T)) = if compatible(C,type(C,M),T) then T else ERRTRM fi .

eq type(C [X : T], (X{0})) = substreduce([shift X] T) .
eq type(C [X : T], (X{s m})) = type(C,(X{m})) .
ceq type(C [X : T], (Y{m})) = type(C, (Y{m})) if not(X == Y) .

eq type(C [X = M], (X{0})) = type(C, M) .
eq type(C [X = M], (X{s m})) = type(C, (X{m})) .
ceq type(C [X = M], (Y{m})) = type(C, (Y{m})) if not(X == Y) .

eq type(C, [X : T] M) = if (type(C [X : T], M) == ERRTRM) then ERRTRM
                        else {X : T} type(C [X : T], M) fi .

eq type(C, [X = N] M) = type(C [X = N], M) .
...
```

Running Example

```
rew type(
     ['U : Type]
     ['eq =
        ['x : 'U]['y : 'U] {'P : 'U -> Prop} ('P 'x) -> ('P 'y) :
        'U -> 'U -> Prop]
     ['eq_is_reflexive =
        ['x : 'U]['P : 'U -> Prop]['H : ('P 'x)] 'H :
        {'x : 'U} ('eq 'x 'x)]
     ['eq_is_transitive =
        ['x : 'U]['y : 'U]['z : 'U]['e1 : ('eq 'x 'y)]['e2 : ('eq 'y 'z)]
        ['P : 'U -> Prop]['h : ('P 'x)]('e2 'P ('e1 'P 'h)) :
        {'x : 'U}{'y : 'U}{'z : 'U} ('eq 'x 'y) -> ('eq 'y 'z) -> ('eq 'x 'z)] ) .
rewrites: 46362
result Context: [`U : Type]
                [`eq : {? : `U{0}} {? : `U{0}} Prop]
                [`eq_is_reflexive : {`x : `U} (`eq `x `x)]
                [`eq_is_transitive : {`x : `U} {`y : `U} {`z : `U}
                                     {? : `eq{0} (`x{0}) (`y{0})}
                                     {? : `eq{0} (`y{0}) (`z{0})}
                                     (`eq{0} (`x{0}) (`z{0}))]
```