EQUATIONAL AND INDUCTIVE REASONING FOR MAUDE IN ATHENA

Mateo Sanabria, Carlos Varela, Camilo Rocha, Nicolás Cardozo

WRLA 2026 — Torino, April 11–12

MAUDE

A language where **equational specifications** are directly executable.

Rewrite rules are first-class computations

ATHENA

A programming language for **proof engineering** and **natural deduction.**

Proofs are first-class citizens

MAUDE

- Order-sorted equational logic
- Rewriting modulo axioms
- High-performance execution
- Limited interactive theorem proving*

ATHENA

- Many-sorted first-order logic
- Natural-deduction proofs
- Structural induction on datatypes
- No native subsorting support

How can we leverage both?

Execute in Maude, prove in Athena.

MAUDE: PEANO NUMBERS

```
 1 fmod PEANO is
 2     sort NzNat Even Nat .
 3     subsorts NzNat Even < Nat .
 4     op zero : -> Even [ctor] .
 5     op s_  : Nat -> NzNat [ctor] .
 6     op _+_ : Nat Nat -> Nat .
 7     vars N M : Nat .
 8     cmb s s N : Even if N : Even .
 9     eq N + zero = N .
10     eq N + s M  = s (N + M) .
11 endfm
```

Sorts/Subsorting

Operations

Conditional membership

Equations

Ctors

ATHENA: PEANO NUMBERS

```
1 datatype Nat := zero | (s Nat)
2 declare plus: [Nat Nat] -> Nat [+]
3 define [n m] := [?n:Nat ?m:Nat]
4 assert  Plus-zero := (forall n . (zero + n) = n)
5 assert* Plus-s    := ((n + (s m)) = (s (n + m)))
```

Datatype Ctors: zero, s Operations Assertions

ATHENA: PROOFS BY INDUCTION

```
1  conclude plus_associative
2  by-induction (forall P Q R .  ((Q + R) + P) = (Q + (R + P))) {
3    zero  => pick-any q r
4             (!chain [((q + r) + zero)
5                      --> (q + r)           [Plus-zero]
6                      <-- (q + (r + zero)) [Plus-zero]])
7    | (s p) => let {ih := (forall ?q ?r .
8                      (?q + ?r) + p = ?q + (?r + p))}
9             pick-any q r
10            (!chain [((q + r) + (s p))
11                     --> (s ((q + r) + p))   [Plus-s]
12                     --> (s (q + (r + p)))   [ih]
13                     <-- (q + (s (r + p)))   [Plus-s]
14                     <-- (q + (r + (s p)))   [Plus-s]])
15 }
```

by-induction: built-in method

✓works because Nat is a datatype

SO... WHAT IS THE CHALLENGE?

A simpler Peano module translates **cleanly**:

```
sort Nat .
op zero : -> Nat [ctor] .
op s_   : Nat -> Nat [ctor] .
```

→

```
datatype Nat := zero | (s Nat)
```

✓ **Nat has no subsorts → becomes a datatype → by-induction works**

But the **full** Peano module has subsorts:

```
sort NzNat Even Nat .
subsorts NzNat Even < Nat .
op zero : -> Even [ctor] .
op s_   : Nat  -> NzNat [ctor]
```

→

✗ **subsorts cannot be directly translated to datatypes → by-induction no longer applies**

MAUDE2ATHENA
KEY CHALLENGES

1. **Subsorting gap:** Maude's order-sorted signatures have no direct counterpart in Athena's many-sorted logic.
2. **Induction loss:** Flattening order-sorted datatypes into many-sorted domains prevents the use of Athena’s built-in inductive reasoning.
3. **Structural axioms:** Maude rewrites modulo ACU. Athena has no built-in support for these axioms

MEMBERSHIP EQUATIONAL LOGIC

`Maude2Athena contemplates functional modules based on equational logic` $\mathcal{E} = (\Sigma, E \cup A)$ `where:`

- Signature $\Sigma = (K, F, S)$: kinds K, function symbols F, sorts S
- Sort poset $(S, \leq)$: subsort relation, connected components are kinds
- Equations E: oriented left-to-right as simplification rules
- Structural axioms A: associativity, commutativity, Unity (ACU)

Assumptions: sort-decreasingness, ground confluence, operational termination, and **sufficient completeness** w.r.t. constructors $\Omega \subseteq \Sigma$.

STRICTLY SENSIBLE SIGNATURES

1. Strongly sensible: argument-compatible overloaded symbols share the same target sort.
2. Maximal argument-bounding: every compatibility class has a unique maximal representative.

Guarantees a linear, unambiguous mapping from order-sorted to many-sorted

> **Precondition** for a well-defined translation

MAUDE2ATHENA

COMPONENTS

tr_S	tr_F	tr_T	tr_E	tr_M	tr_η
Sorts	Operators	Terms	Equations	Memberships	Induction

- tr_S Sorts → **datatype** (no subsorts) or **domain**
- tr_F Maximal representatives + cast symbols + ACU axioms
- tr_T Terms with explicit casts
- tr_E Equations + core equalities
- tr_M Membership predicates (`is_s`)
- tr_η Parametric induction methods

SORT TRANSLATION

A sort *s* becomes a **datatype** if it has constructors and no subsort relations, **domain** otherwise.

$$\mathrm{tr}_S(s) = \begin{cases} \textbf{datatype}(s, \text{constructors}) & \text{if } s \notin \mathrm{dom}(\leq) \cup \mathrm{ran}(\leq) \text{ and } \exists c \in \Omega \\ \textbf{domain}(s) & \text{otherwise} \end{cases}$$

No subsorts → datatype

```
datatype Nat := zero | (s Nat)
```

With subsorts → domains

```
domains NzNat Even Nat
```

✗ domains lose built-in induction — requires tr_η

OPERATORS & CASTS

For each argument-compatibility class, select the **maximal representative**:

```
declare s    : [Nat]      -> NzNat
declare zero : []         -> Even
declare +    : [Nat Nat] -> Nat
```

For each subsort pair $s \leq s'$, generate a **cast symbol**:

```
declare Cast_Even_to_Nat  : [Even]  -> Nat
declare Cast_NzNat_to_Nat : [NzNat] -> Nat
```

ACU axioms become explicit assertions in β

TERMS & EQUATIONS

Implicit subsort coercions become **explicit casts**:

Maude

```
eq N + zero = N .
eq N + s M  =
   s (N + M) .
```

Athena (after tr)

```
assert* ((+ N (Cast_Even_to_Nat zero))
         = N)
assert* ((+ N (Cast_NzNat_to_Nat (s M))) =
         (Cast_NzNat_to_Nat (s (+ N M))))
```

Core equalities ensure cast-path independence: $Cast_{s' \to s''}(Cast_{s \to s'}(x)) = Cast_{s \to s''}(x)$

MEMBERSHIPS

Membership axioms have no native counterpart in Athena — translated to **Boolean predicates**:

Maude

```
cmb s s N : Even
    if N : Even .
```

Athena

```
declare is_even : [Nat] -> Boole
assert* ((is_even N)
         ==> (is_even s s N))
```

Predicate domain is the *kind* — the top supersort in the connected component

FULL PEANO TRANSLATION

Maude input

```
fmod PEANO is
  sort NzNat Even Nat .
  subsorts NzNat Even < Nat .
  op zero : -> Even [ctor] .
  op s_   : Nat -> NzNat [ctor] .
  op _+_  : Nat Nat -> Nat .
  vars N M : Nat .
  cmb s s N : Even
      if N : Even .
  eq N + zero = N .
  eq N + s M = s (N + M) .
endfm
```

Athena output — tr(ℰ)

```
module PEANO {
  domains NzNat Even Nat
  declare s    : [Nat]     -> NzNat
  declare zero : []        -> Even
  declare +    : [Nat Nat] -> Nat
  declare Cast_Even_to_Nat  : [Even]  -> Nat
  declare Cast_NzNat_to_Nat : [NzNat] -> Nat
  define [N M] := [?N:Nat ?M:Nat]
  assert* ((+ N (Cast_Even_to_Nat zero)) = N)
  assert* ((+ N (Cast_NzNat_to_Nat (s M)))
         = (Cast_NzNat_to_Nat (s (+ N M))))
  declare is_even : [Nat] -> Boolean
  assert* ((is_even N) ==> (is_even s s N))
}
```

THE INDUCTION CHALLENGE

Problem: Athena's `by-induction` works **only for datatypes**, but tr_S maps subsorted sorts to *domains* ⇒ native induction is lost

Solution: Generate **parametric primitive methods** (tr_η) that reconstruct induction for domains

1. Compute the *effective constructor set* Ω^+_C per connected component
2. Partition into **base cases** ($I_c = \varnothing$) and **inductive cases** ($I_c \neq \varnothing$)
3. Assemble a primitive method with nested `check` / `holds?`

PRIMITIVE INDUCTION METHOD

```
primitive-method (nat-induction property) :=
let {
    basis := (property (Cast_Even_to_Nat zero));
    ic    := (forall x
                (if (property x)
                    (property (Cast_NzNat_to_Nat (s x)))))
  }
check {
    (holds? basis) =>
      check { (holds? ic) =>
                 (forall x (property x))
            | else => (error "Inductive step fails.") }
    | else => (error "Basis step fails.") }
```

primitive-method: takes a predicate

basis + ic: the two obligations

check / holds?: sequential verification

✓ Restores structural induction for the `Nat` *domain* — generated automatically by tr_{η}

CASE STUDY: TOY COMPILER

A compiler for arithmetic expressions on a Stack Machine — three sorts, heavy use of subsorting and structural axioms:

`Exp`
source expressions
(Int < Exp)

`Instr/Program`
instruction set
(Instr < Program)

`Stack`
runtime stack
(datatype)

Core correctness property proven in Athena:

```
forall e p s . (exec (++ (compile e) p) s)
             = (exec p (:: (I e) s))
```

✗ **Exp is a domain — no native induction**

✓ **exp-induction generated by tr_{η}**

📦 Full specification + proofs: `github.com/FLAGlab/Maude2Athena`

CONTRIBUTIONS

1 Semantics-preserving translation from Maude's membership equational logic to Athena's many-sorted first-order logic

2 Parametric induction methods (tr_{η}) that reconstruct structural induction over translated domains

3 First implementation grounded in Li's et al strictly sensible order-sorted translation — validated on a compiler case study

FUTURE WORK

→ Map Maude's built-in modules directly to Athena's native domains for better modularity and automation

→ Extend to rewrite rules — enabling verification of concurrent and distributed systems via Talcott's Actor theories

→ Bridge model checking and theorem proving — leveraging both Maude's execution and Athena's deductive power