CICM 2017

Formalizing Mathematical Knowledge as a Biform Theory Graph: A Case Study

Jacques Carette and William M. Farmer

Department of Computing and Software
McMaster University

18 July 2017

MathScheme Project [CFO11]

- A long-term project at McMaster University lead by Jacques Carette and William Farmer.
- Its objective is to develop a new approach to mechanized mathematics in which axiomatic and algorithmic mathematics are tightly integrated.
- Two key ideas:
    1. Little theories method.
    2. Biform theories.

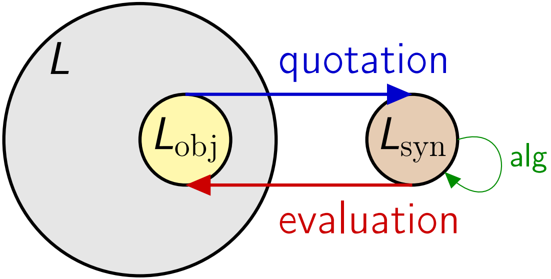
Little Theories Method [FGT92]

- A complex body of mathematical knowledge is encoded by the little theories method as a theory graph [Kol14] in which:
  1. The nodes are theories.
  2. The edges are theory morphisms, meaning-preserving mappings from one theory to another.
- The theory morphisms are information conduits that enable definitions and theorems to be transported between theories.
- A theory graph enables formalization at the most convenient level of abstraction using the most convenient vocabulary.
- The tiny theories method is a refinement in which each theory is obtained from another theory by the addition of a single concept.

Biform Theories [FM03,CF08]

- A biform theory is a triple $T = (L, \Pi, \Gamma)$ where:
  1. L is a language of some underlying logic that is generated from a set of symbols (e.g., types and constants).
  2. The expressions in L denote mathematical values that include syntactic values representing the expressions of L.
  3. Π is a set of transformers that implement functions on the expressions of L and are represented by symbols of L.
  4. Γ is a set of axioms (formulas of L) that express properties about the symbols (and thus about the transformers) of L.
- Transformers may be specified in T but implemented externally.
- T is an axiomatic theory if Π is empty and an algorithmic theory if Γ is empty.
- An implementation of a biform theory requires a metareasoning infrastructure with reflection using a local or global approach.

Metareasoning Infrastructure

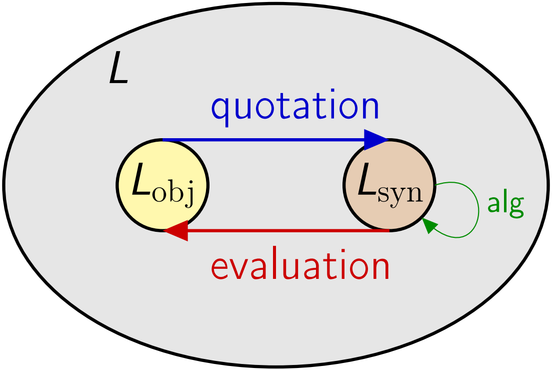


L: language of the logic.

L_{obj}: language on which to perform computation.

L_{syn}: language of syntactic values for L_{obj} (in the metalogic).

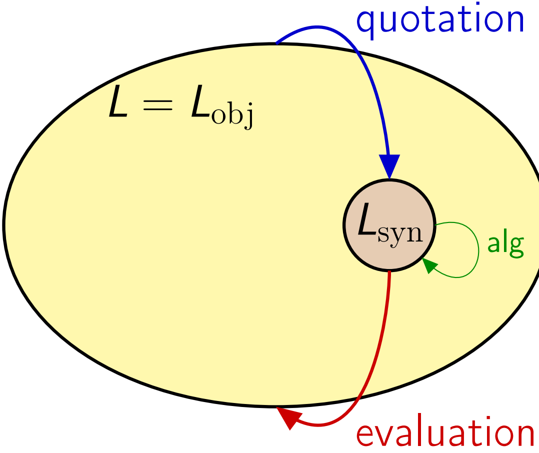
Metareasoning with Local Reflection



Advantages: (1) L_{syn} resides in the logic and (2) the approach is implementable in many logics.

Disadvantage: Does not scale up — many of these infrastructures are needed.

Metareasoning with Global Reflection



Advantages: (1) L_{syn} resides in the logic and
(2) this approach scales up — only one infrastructure is needed.
Disadvantage: Requires a nontrivial modification of the logic.

MathScheme Framework

1. Biform theories are used to combine axiomatic and algorithmic mathematical knowledge.
2. Biform theories are expressed in a formal logic that supports undefinedness and metareasoning with global reflection.
   - Undefinedness: pf, pf*, lutins, nbg*, stmm, $\mathcal{Q}_0^{\mathrm{u}}$.
   - Quotation and evaluation: Chiron, $\mathcal{Q}_0^{\mathrm{uqe}}$, $\mathrm{ctt}_{\mathrm{qe}}$, $\mathrm{ctt}_{\mathrm{uqe}}$.
3. A body of mathematical knowledge is encoded as a graph of biform theories using the little/tiny theories method.
4. Algorithms are implemented as generic programs that are specialized using code generation.

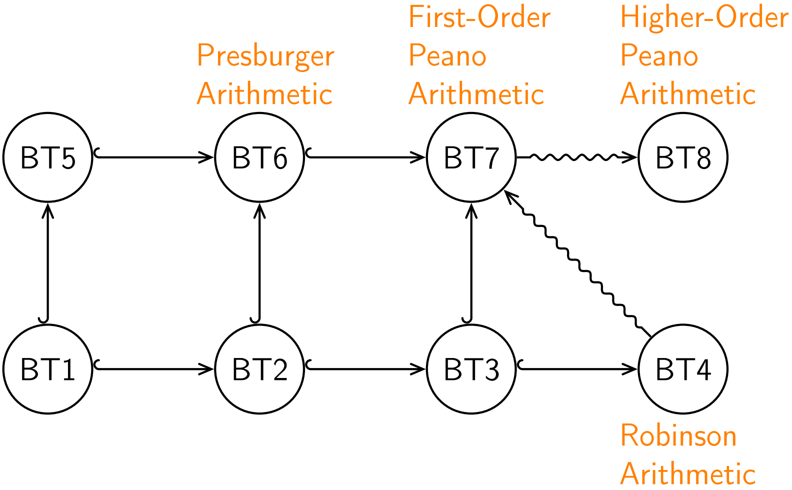
Open Question

- Can the little/tiny theories method be applied to biform theories?
- Test case: A graph of eight biform theories that encode natural number arithmetic and include a variety of transformers.
- We describe and compare two formalizations of the test case:
  1. In $\mathrm{ctt}_{\mathrm{uqe}}$ [Far17] using global reflection.
     - $\mathrm{ctt}_{\mathrm{uqe}}$ is a variant of $\mathrm{ctt}_{\mathrm{qe}}$ [Far16,Far16a], a version of Church's type theory with global quotation and evaluation operators.
     - $\mathrm{ctt}_{\mathrm{uqe}}$ includes a notion of a theory morphism.
     - $\mathrm{ctt}_{\mathrm{uqe}}$ and $\mathrm{ctt}_{\mathrm{qe}}$ are not yet implemented.
  2. In Agda using local reflection.
     - Agda is an implemented, dependently typed programming language.
     - Work is being done to add global reflection to Agda [WS12].

Test Case: The Biform Theories

	Logic	Constants	Induction	Transformers	Decidability
BT1	FOL	$0, S$		IS-FO-BT1	undecidable*
BT2	FOL	$0, S, +$		IS-FO-BT2, BPLUS	undecidable*
BT3	FOL	$0, S, +, *$		IS-FO-BT3, BTIMES	undecidable*
BT4	FOL	$0, S, +, *$	$x = 0 \vee \exists\, y \,.\, S(y) = x$		essentially undecidable
BT5	FOL	$0, S$	induction schema	BT5-DEC-PROC	decidable
BT6	FOL	$0, S, +$	induction schema	BT6-DEC-PROC	decidable
BT7	FOL	$0, S, +, *$	induction schema		essentially undecidable
BT8	STT	$0, S$	induction axiom		essentially undecidable

Test Case: The Biform Theory Graph



The $\hookrightarrow$ arrows denote theory inclusions.

The $\rightsquigarrow$ arrows denote theory morphisms that are not inclusions.

BT6 (Presburger Arithmetic) in $\mathrm{ctt}_{\mathrm{qe}}$ [1/5]

Primitive Base Types

1. ι (type of natural numbers).

Primitive Constants

1. 0_ι.
2. $S_{\iota \rightarrow \iota}$.
3. $+_{\iota \rightarrow \iota \rightarrow \iota}$ (infix).
4. $\mathsf{BPLUS}_{\epsilon \rightarrow \epsilon \rightarrow \epsilon}$ (infix).
6. $\mathsf{BT6\text{-}DEC\text{-}PROC}_{\epsilon \rightarrow \epsilon}$.

Defined Constants (selected)

1. $1_\iota = S\, 0_\iota$.

BT6 (Presburger Arithmetic) in $\mathrm{ctt}_{\mathrm{qe}}$ [2/5]

3. $\mathsf{bnat}_{\iota \to \iota \to \iota} = \lambda\, x_\iota \,.\, \lambda\, y_\iota \,.\, ((x_\iota + x_\iota) + y_\iota)$.
   Notational definition:

   $$\begin{array}{ll} (0)_2 = \mathsf{bnat}\, 0_\iota\, 0_\iota. & \\ (1)_2 = \mathsf{bnat}\, 0_\iota\, 1_\iota. & \\ (a_1 \cdots a_n 0)_2 = \mathsf{bnat}\, (a_1 \cdots a_n)_2\, 0_\iota & \text{where each } a_i \text{ is 0 or 1.} \\ (a_1 \cdots a_n 1)_2 = \mathsf{bnat}\, (a_1 \cdots a_n)_2\, 1_\iota & \text{where each } a_i \text{ is 0 or 1.} \end{array}$$

4. $\mathsf{is\text{-}bnum}_{\epsilon \to o} = \mathrm{I}\, f_{\epsilon \to o} \,.\, \forall\, u_\epsilon \,.\, (f_{\epsilon \to \epsilon}\, u_\epsilon \equiv$
   $\exists\, v_\epsilon \,.\, \exists\, w_\epsilon \,.\, (u_\epsilon = \ulcorner \mathsf{bnat}\, \lfloor v_\epsilon \rfloor\, \lfloor w_\epsilon \rfloor \urcorner \wedge$
   $(v_\epsilon = \ulcorner 0_\iota \urcorner \vee f_{\epsilon \to \epsilon}\, v_\epsilon) \wedge (w_\epsilon = \ulcorner 0_\iota \urcorner \vee w_\epsilon = \ulcorner 1_\iota \urcorner)))$.
5. $\mathsf{IS\text{-}FO\text{-}BT2}_{\epsilon \to \epsilon} = \lambda\, x_\epsilon \,.\, \mathbf{B}_\epsilon$ where $\mathbf{B}_\epsilon$ is a complex expression such that $(\lambda\, x_\epsilon \,.\, \mathbf{B}_\epsilon) \ulcorner \mathbf{A}_\alpha \urcorner$ equals $\ulcorner T_o \urcorner$ [$\ulcorner F_o \urcorner$] if $\mathbf{A}_\alpha$ is [not] a term or formula of first-order logic whose variables are of type ι and nonlogical constants are members of $\{0_\iota, S_{\iota \to \iota}, +_{\iota \to \iota \to \iota}\}$.
7. $\mathsf{IS\text{-}FO\text{-}BT2\text{-}ABS}_{\epsilon \to \epsilon} =$
   $\lambda\, x_\epsilon \,.\, (\mathsf{if}\, (\mathsf{is\text{-}abs}_{\epsilon \to o}\, x_\epsilon)\, (\mathsf{IS\text{-}FO\text{-}BT2}_{\epsilon \to \epsilon}\, (\mathsf{abs\text{-}body}_{\epsilon \to \epsilon}\, x_\epsilon))\, \ulcorner F_o \urcorner)$.

BT6 (Presburger Arithmetic) in $\mathrm{ctt}_{\mathrm{qe}}$ [3/5]

Axioms

1. $S\,x_\iota \neq 0_\iota$.
2. $S\,x_\iota = S\,y_\iota \supset x_\iota = y_\iota$.
3. $x_\iota + 0_\iota = x_\iota$.
4. $x_\iota + S\,y_\iota = S\,(x_\iota + y_\iota)$.
5. is-bnum $u_\epsilon \supset u_\epsilon$ BPLUS $\ulcorner(0)_2\urcorner = u_\epsilon$.

$\vdots$

15. (is-bnum $u_\epsilon \wedge$ is-bnum $v_\epsilon) \supset$
$\ulcorner\mathrm{bnat}\,\lfloor u_\epsilon \rfloor\, 1_\iota\urcorner$ BPLUS $\ulcorner\mathrm{bnat}\,\lfloor v_\epsilon\rfloor\, 1_\iota\urcorner =$
$\ulcorner\mathrm{bnat}\,\lfloor(u_\epsilon$ BPLUS $v_\epsilon)$ BPLUS $\ulcorner(1)_2\urcorner\rfloor\, 0_\iota\urcorner$.

29. Induction Schema for S and $+$
$\forall\, f_\epsilon\,.\,((\mathrm{is\text{-}expr}^{\iota\to o}_{\epsilon\to o}\, f_\epsilon \wedge [\![\mathrm{IS\text{-}FO\text{-}BT2\text{-}ABS}_{\epsilon\to\epsilon}\, f_\epsilon]\!]_o) \supset$
$(([\![f_\epsilon]\!]_{\iota\to o}\, 0_\iota \wedge (\forall\, x_\iota\,.\,[\![f_\epsilon]\!]_{\iota\to o}\, x_\iota \supset [\![f_\epsilon]\!]_{\iota\to o}\,(\mathrm{S}\,x_\iota))) \supset$
$\forall\, x_\iota\,.\,[\![f_\epsilon]\!]_{\iota\to o}\, x_\iota))$.

BT6 (Presburger Arithmetic) in $\mathrm{ctt}_{\mathrm{qe}}$ [4/5]

29. Meaning formula for $\mathrm{BT6\text{-}DEC\text{-}PROC}_{\epsilon\to\epsilon}$.
    $\forall\, u_\epsilon \,.\, ((\text{is-expr}^{o}_{\epsilon\to o}\, u_\epsilon \wedge \text{is-closed}_{\epsilon\to o}\, u_\epsilon \wedge [\![\mathrm{IS\text{-}FO\text{-}BT2}_{\epsilon\to\epsilon}\, u_\epsilon]\!]_o) \supset$
    $((\mathrm{BT6\text{-}DEC\text{-}PROC}_{\epsilon\to\epsilon}\, u_\epsilon = \ulcorner T_o \urcorner \vee$
    $\mathrm{BT6\text{-}DEC\text{-}PROC}_{\epsilon\to\epsilon}\, u_\epsilon = \ulcorner F_o \urcorner) \wedge$
    $[\![\mathrm{BT6\text{-}DEC\text{-}PROC}_{\epsilon\to\epsilon}\, u_\epsilon]\!]_o = [\![u_\epsilon]\!]_o))$.

Transformers

3. π_3 for $\mathrm{BPLUS}_{\epsilon\to\epsilon\to\epsilon}$ is an efficient program that satisfies Axioms 5–15.
4. π_4 for $\mathrm{BPLUS}_{\epsilon\to\epsilon\to\epsilon}$ uses Axioms 5–15 as conditional rewrite rules.
5. π_5 for $\mathrm{IS\text{-}FO\text{-}BT2}_{\epsilon\to\epsilon}$ is an efficient program that accesses the data stored in the data structures that represent expressions.
6. π_6 for $\mathrm{IS\text{-}FO\text{-}BT2}_{\epsilon\to\epsilon}$ uses the definition of $\mathrm{IS\text{-}FO\text{-}BT2}_{\epsilon\to\epsilon}$.
14. π_{14} for $\mathrm{BT6\text{-}DEC\text{-}PROC}_{\epsilon\to\epsilon\to\epsilon}$ is an efficient decision procedure that satisfies Axiom 30.

BT6 (Presburger Arithmetic) in $\mathrm{ctt}_{\mathrm{qe}}$ [5/5]

15. π_{15} for IS-FO-BT2-ABS$_{\epsilon\to\epsilon}$ is an efficient program that accesses the data stored in the data structures that represent expressions.
16. π_{16} for IS-FO-BT2-ABS$_{\epsilon\to\epsilon}$ uses the definition of IS-FO-BT2-ABS$_{\epsilon\to\epsilon}$.

Theorems (selected)

3. Meaning formula for BPLUS$_{\epsilon\to\epsilon\to\epsilon}$
$$\forall\, u_\epsilon \,.\, \forall\, v_\epsilon \,.\, ((\text{is-bnum}\, u_\epsilon \wedge \text{is-bnum}\, v_\epsilon) \supset (\text{is-bnum}\, (u_\epsilon \text{ BPLUS } v_\epsilon) \wedge (\llbracket u_\epsilon \text{ BPLUS } v_\epsilon \rrbracket_\iota = \llbracket u_\epsilon \rrbracket_\iota + \llbracket v_\epsilon \rrbracket_\iota))).$$

BT1 in Agda

```
record BT₁ : Set₁ where
  field
    nat : Set₀
    Z : nat
    S : nat → nat
    S≠Z : ∀ x → ¬ (S x ≡ Z)
    inj : ∀ x y → S x ≡ S y → x ≡ y

  One : nat
  One = S Z

  ⟦_⟧₁ : ℕ → nat
  ⟦ 0 ⟧₁ = Z
  ⟦ suc x ⟧₁ = S ⟦ x ⟧₁
```

Numerals in Agda

```
data BinDigit : Set where zero one : BinDigit
data BNum : Set where
   bn : {n : ℕ} → Vec BinDigit (suc n) → BNum

« : BNum → BNum
« (bn l) = bn (zero l)

+1 : BNum → BNum
+1 (bn (zero l)) = bn (one l)
+1 (bn (one [])) = bn (zero one [])
+1 (bn (one x l)) = « (+1 (bn (x l)))

bplus : BNum → BNum → BNum
bplus (bn {0} (zero [])) y = y
bplus (bn {0} (one [])) y = +1 y
bplus (bn {suc n} (d0 l0)) (bn {ℕ.zero} (zero [])) = bn (d0 l0)
bplus (bn {suc n} (d0 l0)) (bn {ℕ.zero} (one [])) = +1 (bn (d0 l0))
bplus (bn {suc n} (zero l0)) (bn {suc m} (zero l1)) = « (bplus (bn l0) (bn l1))
bplus (bn {suc n} (one l0)) (bn {suc m} (zero l1)) = +1 (« (bplus (bn l0) (bn l1)))
bplus (bn {suc n} (zero l0)) (bn {suc m} (one l1)) = +1 (« (bplus (bn l0) (bn l1)))
bplus (bn {suc n} (one l0)) (bn {suc m} (one l1)) =
   +1 (+1 (« (bplus (bn l0) (bn l1))))
```

BT2 in Agda

```
record BT₂ (t1 : BT₁) : Set where
  open BT₁ t1 public
  field
    _+_ : nat → nat → nat
    right-0 : ∀ x → x + Z ≡ x
    x+Sy≡Sx+y : ∀ x y → x + S y ≡ S (x + y)

  bnat : nat → nat → nat
  bnat x y = (x + x) + y

  dig-to-nat : BinDigit → nat
  dig-to-nat zero = Z
  dig-to-nat one = S Z
  unroll : {n : ℕ} → Vec BinDigit n → nat
  unroll [] = Z
  unroll (x l) = bnat (unroll l) (dig-to-nat x)

  ⟦_⟧₂ : BNum → nat
  ⟦ bn (x l) ⟧₂ = bnat (unroll l) (dig-to-nat x)
```

Language infrastructure

```
record GroundLanguage (T : Set₀) : Set₁ where
  field
    Lang : DT → Set₀
    value : {V : DT} → Lang V → (Carrier V → T) → T

record LogicOverL (T : Set₀) (L : GroundLanguage T) : Set₁ where
  field
    Logic : DT → Set₀
    ⟦_⟧_ : ∀ {V} → Logic V → (Carrier V → T) → Set₀

module FOL {T : Set₀} (L : GroundLanguage T) where

  data FOL (V : DT) : Set where
    tt : FOL V
    ff : FOL V
    _and_ : FOL V → FOL V → FOL V
    _or_ : FOL V → FOL V → FOL V
    not : FOL V → FOL V
    _⊃_ : FOL V → FOL V → FOL V
    _==_ : Lang V → Lang V → FOL V
    all : Carrier V → FOL V → FOL V
    exist : Carrier V → FOL V → FOL V
```

Logic over a Language, back into Agda

```
LoL-FOL : LogicOverL T L
LoL-FOL = record { Logic = FOL ; ⟦_⟧_ = interp }
  where
    interp : {Var : DT} → FOL Var → (Carrier Var → T) → Set₀
    interp tt env = ⊤
    interp ff env = ⊥
    interp (e and f) env = interp e env × interp f env
    interp (e or f) env = ¬ ¬ (interp e env ⊎ interp f env)
    interp (not e) env = ¬ (interp e env)
    interp (e ⊃ f) env = (interp e env) → (interp f env)
    interp (x == y) env = value x env ≡ value y env
    interp {V} (all x p) env  = ∀ z → interp p (override {V} env x z)
    interp {V} (exist x p) env = ¬ ¬ (Σ T (λ t → interp p (override {V} env x t)))
```

BT6 (Presburger Arithmetic) in Agda

```
record BT₆ {t₁ : BT₁} (t₂ : BT₂ t₁) (t₅ : BT₅ t₁) : Set₁ where
  open VarLangs using (XV; x)
  open DecSetoid using (Carrier)
  open BT₂ t₂ public
  open fo₂ using (FOL; tt; ff; LoL-FOL; _and_; all)
  open LogicOverL LoL-FOL

  field
    induct : (e : FOL XV) →
      ⟦ e ⟧ (λ { x → ⟦ 0 ⟧₁ }) →
      (∀ y → ⟦ e ⟧ (λ {x → y}) → ⟦ e ⟧ (λ {x → S y})) →
      ∀ y → ⟦ e ⟧ (λ {x → y})
  postulate
    decide : ∀ {W} → (Carrier W → nat) → FOL W → FOL NoVars
    meaning-decide : {W : DT} (env : Carrier W → nat) → (env′ : ⊥ → nat) →
      (e : FOL W) →
      let res = decide env e in
      (res ≡ tt ⊎ res ≡ ff) × (⟦ e ⟧ env) ≃ (⟦ res ⟧ env′)
```

Conclusion

- We have proposed a biform theory graph test case encoding natural number arithmetic and including various transformers.

Conclusion

- We have proposed a biform theory graph test case encoding natural number arithmetic and including various transformers.
- We have formalized the test case in (1) $\mathrm{ctt}_{\mathrm{uqe}}$ using global reflection and (2) Agda using the local reflection.
    - We have obtain substantial partial formalizations that indicate full formalizations could be obtain with additional work.

Conclusion

- We have proposed a biform theory graph test case encoding natural number arithmetic and including various transformers.
- We have formalized the test case in (1) $\mathrm{ctt}_{\mathrm{uqe}}$ using global reflection and (2) Agda using the local reflection.
  - We have obtain substantial partial formalizations that indicate full formalizations could be obtain with additional work.
- Our results show that the global approach has a significant advantage over the local approach —

  since the local approach requires a separate infrastructure for each set of expressions manipulated by a transformer.

Conclusion

- We have proposed a biform theory graph test case encoding natural number arithmetic and including various transformers.
- We have formalized the test case in (1) $\mathbf{ctt}_{\mathrm{uqe}}$ using global reflection and (2) Agda using the local reflection.
  - We have obtain substantial partial formalizations that indicate full formalizations could be obtain with additional work.
- Our results show that the global approach has a significant advantage over the local approach —

  since the local approach requires a separate infrastructure for each set of expressions manipulated by a transformer.
- We recommend that future research be directed to making the global approach practical for formalizing biform theories.

References

- [WS12] P. van der Walt and W. Swierstra, “Engineering proof by reflection in Agda”, in: *Implementation and Application of Functional Languages*, LNCS 8241:157–173, 2012.
- [CF08] J. Carette and W. M. Farmer, “High-level theories”, in: *Intelligent Computer Mathematics*, LNCS 5144:232–245, 2008.
- [CFO11] J. Carette, W. M. Farmer, and R. O’Connor, “MathScheme: Project description”, in: *Intelligent Computer Mathematics*, LNCS 6824:287–288, 2011.
- [Far16] W. M. Farmer, “Incorporating quotation and evaluation into Church’s type theory: Syntax and semantics”, in: *Intelligent Computer Mathematics*, LNCS 9791:83–98, 2016
- [Far16a] W. M. Farmer, “Incorporating quotation and evaluation into Church’s type theory”, in: CoRR, abs/1612.02785 (72 pp.), 2016.
- [Far17] W. M. Farmer, “Theory morphisms in Church’s type theory with quotation and evaluation”, in: *Intelligent Computer Mathematics*, LNCS 10383, 16 pp., 2017.
- [FGT92] W. M. Farmer, J. D. Guttman, and F. J. Thayer, “Little theories”, in: *Automated Deduction — CADE-11*, LNCS 607:567–581, 1992.
- [FM03] W. M. Farmer and M. von Mohrenschildt, “An overview of a Formal Framework for Managing Mathematics”, Annals of Mathematics and Artificial Intelligence 38:165–191, 2003.
- [Kol14] M. Kohlhase, “Mathematical knowledge management: Transcending the one-brain-barrier with theory graphs”, EMS Newletter 92:22–27, 2014.