Modular dependent induction in Coq, Mendler-style

Paolo Torrini

Dept. of Computer Science, KU Leuven*

ITP'16, Nancy, 22.08.2016

* The Author left the Institution in April 2016

motivation: cost-effective theorem proving

- modularity in specifications and proofs
  - component-based definitions:
    enabling partiality
  - extensibility:
    reuse when code is extended,
    no need for reimplementation
- low-cost: closeness with conventional ones
- goal: better scalability in programming language semantics

expression problem in functional programming

- conventional inductive datatypes
  - associated with a fixed set of constructors
  - inherently not modular
- extending a conventional datatype requires
  - defining a new datatype
  - reimplementing functions, remaking proofs

solving the expression problem: modular datatypes (MDTs)

- non-total functional languages (e.g. Haskell):
  datatypes á la carte
  [Swierstra JFP'08]
  - based on initial algebra semantics of inductive types

- Coq (totality required):
  meta-theory á la carte (MTC/3MT)
  [Delaware, Oliveira, Schrijvers POPL'13]
  - based on higher-order encodings of initial semantics
  - inductive reasoning by algebraic properties
  - significant boilerplate

Mendler-style modular induction

- applying Mendler-style induction to modular datatypes
  [Torrini, Schrijvers FICS'15]
  - Mendler-style higher-order encodings
  - type-directed approach
  - restriction to non-dependent induction
    (corresponding to iteration)

- extending to the general case (dependent induction)
  current work [Torrini ITP'16]
  - integration of Mendler-style induction (type-directed)
    with minimal use of MTC-style induction (algebraic)
  - structural induction without restrictions

modular datatypes

- MDT definition
  - signature functor *F*
    non-recursive datatype
    fmap satisfying functor laws
  - recursive datatype Fix *F*
    using a type-level fixpoint operator
- extensibility: functors can be composed by *coproduct* (+)
- structurally recursive functions: defined by *fold* of algebras on fixpoints

example: definition of an arithmetic language – conventional datatypes –

terms (natural literals, sums):

$$\mathbf{Trm} =_{dt} \mathit{Lit}\ (\mathbf{Nat}) \mid \mathit{Add}\ (\mathbf{Trm},\ \mathbf{Trm})$$

values (integers):

$$\mathbf{Val} =_{dt} \mathit{Val}\ (\mathit{vv} :\ \mathbf{Nat})$$

example: language definition with MDT

modular datatype, monolithic functor:

$$\mathbf{Trm_F}\ C\ =_{dt}\ \mathit{Lit}\ (\mathbf{Nat})\ \mid\ \mathit{Add}\ (C,\ C)$$

recursive datatype as fixpoint of the functor:

$$\mathbf{Trm}\ :=\ \mathbf{Fix\ Trm_F}$$

$$\mathbf{Trm}\quad\cong\quad\mathbf{Trm}$$

example: language definition with MDT

modular datatype, composite functor (using coproduct):

$$\mathbf{Trm_{F1}}\ C =_{dt} \mathit{Lit}\ (\mathbf{Nat})$$

$$\mathbf{Trm_{F2}}\ C =_{dt} \mathit{Add}\ (C,\ C)$$

$$\mathbf{Trm_F} := \mathbf{Trm_{F1}} + \mathbf{Trm_{F2}}$$

recursive datatype as fixpoint of the functor:

$$\mathbf{Trm} := \mathbf{Fix}\ \mathbf{Trm_F}$$

$$\mathbf{Trm} \cong \mathbf{Trm}$$

example: evaluation function – conventional definition –

$$\mathbf{Trm} =_{dt} \mathit{Lit}\ (\mathbf{Nat}) \mid \mathit{Add}\ (\mathbf{Trm},\ \mathbf{Trm})$$

$$\mathbf{Val} =_{dt} \mathit{Val}\ (\mathit{vv} :\ \mathbf{Nat})$$

$$\begin{aligned}
&\mathit{eval} : \mathbf{Trm} \to \mathbf{Val} \\
&\mathit{eval}\ (\mathit{Lit}\ n) := \mathit{Val}\ n \\
&\mathit{eval}\ (\mathit{Add}\ (e_1, e_2)) := \mathit{Val}\ (\mathit{vv}\ (\mathit{eval}\ e_1) + \mathit{vv}\ (\mathit{eval}\ e_2))
\end{aligned}$$

example: evaluation function for MDT (1) Trm_F-algebra with carrier Val

$$\mathbf{Trm_F}\ C\ =_{dt}\ Lit\ (\mathbf{Nat})\ \mid\ Add\ (C,\ C)$$

$$eval_C\ :\ \mathbf{Trm_F\ Val} \rightarrow \mathbf{Val}$$

$$eval_C\ \ (Lit\ n)\ :=\ Val\ n$$

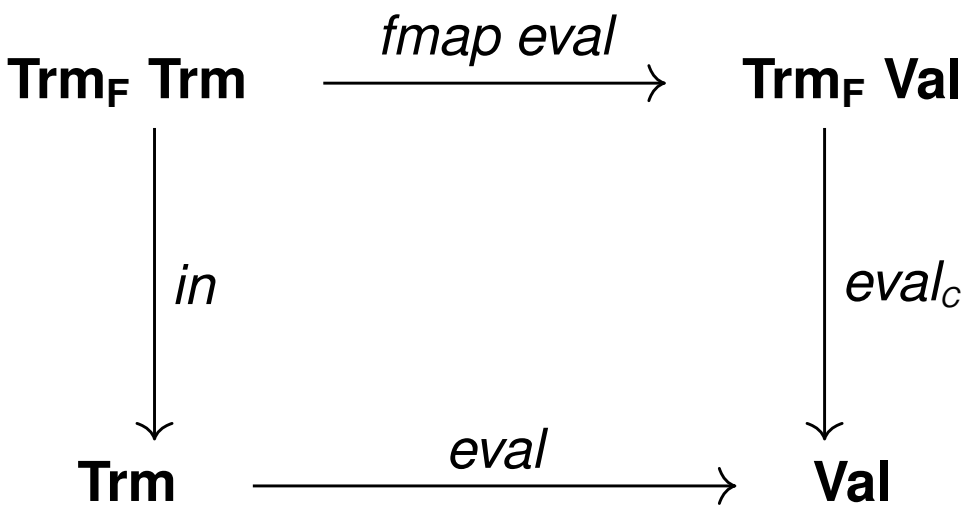
$$eval_C\ \ (Add\ (u_1, u_2))\ :=\ Val\ (vv\ u_1\ +\ vv\ u_2)$$

example: evaluation function for MDT (2) recursion by folding

$$\mathbf{Trm} := \mathbf{Fix}\ \mathbf{Trm_F}$$

$$\mathit{eval} : \mathbf{Trm} \to \mathbf{Val}$$

$$\mathit{eval} := \mathit{fold}\ \mathbf{Trm_F}\ \mathbf{Val}\ \mathit{eval_C}$$



example: evaluation function for MDT
(3) Mendler Trm_F-algebra with carrier Val

$$\mathbf{Trm_F}\ C\ =_{dt}\ Lit\ (\mathbf{Nat})\ \mid\ Add\ (C,\ C)$$

$$eval_M\ :\quad \forall A.\ (A \to \mathbf{Val}) \to (\mathbf{Trm_F}\ A \to \mathbf{Val})$$

$$eval_M\quad A\quad rc\quad (Lit\ n)\qquad\qquad :=\ Val\ n$$

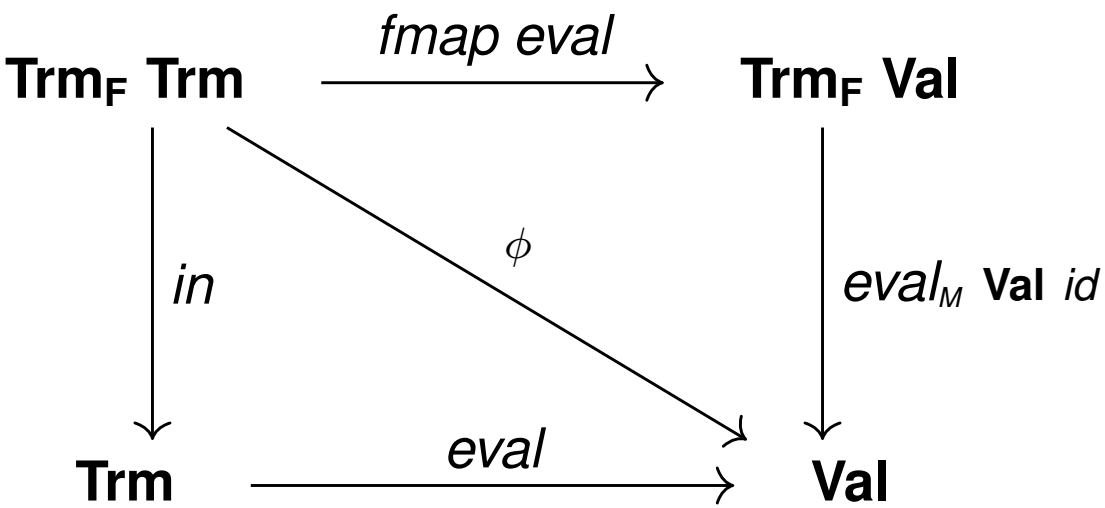
$$eval_M\quad A\quad rc\quad (Add\ (u_1, u_2))\ :=\ Val\ (vv\ (rc\ u_1)\ +\ vv\ (rc\ u_2))$$

example: evaluation function for MDT
(4) recursion by folding (Mendler-style)

$$\mathbf{Trm} := \mathbf{Fix}\ \mathbf{Trm_F}$$

$$\mathit{eval} : \mathbf{Trm} \to \mathbf{Val}$$

$$\mathit{eval} := \mathit{fold}\ \mathbf{Trm_F}\ \mathbf{Val}\ \mathit{eval}_M$$



where $\phi := \mathit{eval}_M\ \mathbf{Trm}\ \mathit{eval}$

critical notions: *Fix* and *fold*

in Haskell: no guarantee of totality / termination

$$\mathbf{Fix}\ F \ =_{dt}\ \mathit{In}\ (\mathit{out} : F\ (\mathbf{Fix}\ \ \mathrm{F}))$$

$$\mathit{fold}\ f\ x\ :=\ f\ (\mathit{fmap}\ (\mathit{fold}\ f)\ (\mathit{out}\ x))$$

critical notions: *Fix* and *fold*

in a theorem prover: termination needed for consistency

strictly positive datatypes, structurally recursive definitions

(!) $\mathbf{Fix}\ F =_{dt} In\ (out : F\ (\mathbf{Fix}\ F))$

non-positive occurrence of **Fix**

(!) $fold\ f\ x := f\ (fmap\ (fold\ f)\ (out\ x))$

not structurally recursive

modular reasoning in Coq

- encoding MDTs
  - direct encoding of signature functors
  - higher-order, eliminative encoding of fixed points: Church-style (conventional) or equiv. Mendler-style
  - impredicative sets needed
- close-up problem: eliminative definitions complicate induction
- background problem: semantic soundness (*fold* uniqueness)
- dealing with inductive reasoning:
  - using Mendler algebras, Mendler-style induction can be used for non-dependent induction
  - MTC/3MT: general solution by algebraic reasoning, using *fold* uniqueness
  - integrating the two techniques

algebra types

- endofunctor F on sets, C set
- F-algebras with carrier C
- type of conventional Church algebras

$$\mathbf{Alg^C}\ F\ C := F\ C \to C$$

  - semantically: a morphism on sets

- type of Mendler algebras

$$\mathbf{Alg^M}\ F\ C := \forall A.\ (A \to C) \to F\ A \to C$$

  - semantically: a function between morphisms
  - A: approximates recursive call argument type
    (restriction: not used elsewhere, not further analysed)
  - $A \to C$: iterative call type

fixpoint and fold – eliminative encoding

Church encoding

- type-level fixpoint operator – not a constructor

$$\mathbf{Fix}^{\mathbf{C}}\ F\ :=\ \forall A.\ \mathbf{Alg}^{\mathbf{C}}\ F\ A \to A$$

- *fold* as application of a fixpoint

$$\mathit{fold}^{C}\ F\ C : \mathbf{Alg}^{\mathbf{C}}\ F\ C \to \mathbf{Fix}^{\mathbf{C}}\ F \to C$$

$$\mathit{fold}^{C}\ F\ C\ \mathit{alg}\ x\ :=\ x\ \mathit{alg}$$

- defined functions – not constructors

$$\mathit{in}^{C}\ F :\ F(\mathbf{Fix}^{\mathbf{C}}\ F) \to \mathbf{Fix}^{\mathbf{C}}\ F$$

$$\mathit{out}^{C}\ F :\ \mathbf{Fix}^{\mathbf{C}}\ F \to F(\mathbf{Fix}^{\mathbf{C}}\ F)$$

fixpoint and fold – eliminative encoding

Mendler encoding

- type-level fixpoint operator – not a datatype

$$\mathbf{Fix^M}\ F\ :=\ \forall A.\ \mathbf{Alg^M}\ F\ A \to A$$

- *fold* as application of a fixpoint

$$\mathit{fold}^M\ F\ C : \mathbf{Alg^M}\ F\ C \to \mathbf{Fix^M}\ F \to C$$

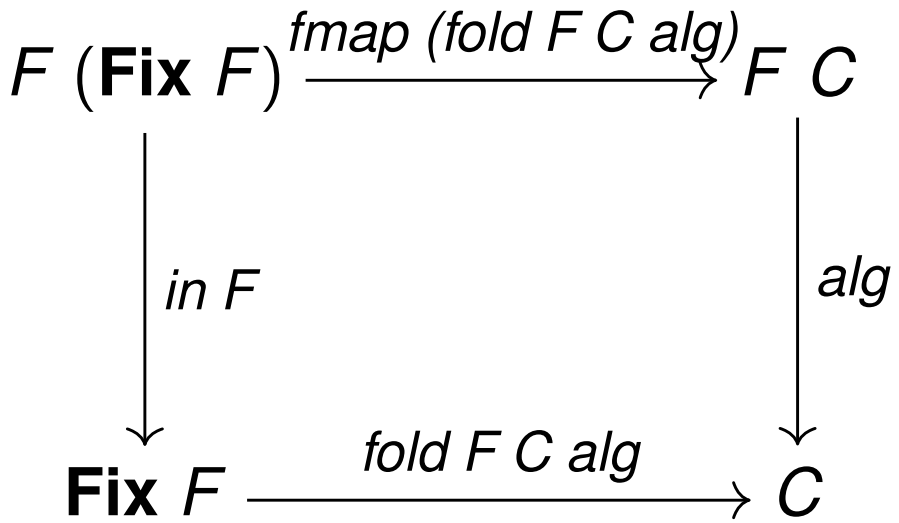
$$\mathit{fold}^M\ F\ C\ \mathit{alg}\ x\ :=\ x\ \mathit{alg}$$

- defined functions – not constructors

$$\mathit{in}^M\ F :\ F(\mathbf{Fix^M}\ F) \to \mathbf{Fix^M}\ F$$

$$\mathit{out}^M\ F :\ \mathbf{Fix^M}\ F \to F(\mathbf{Fix^M}\ F)$$

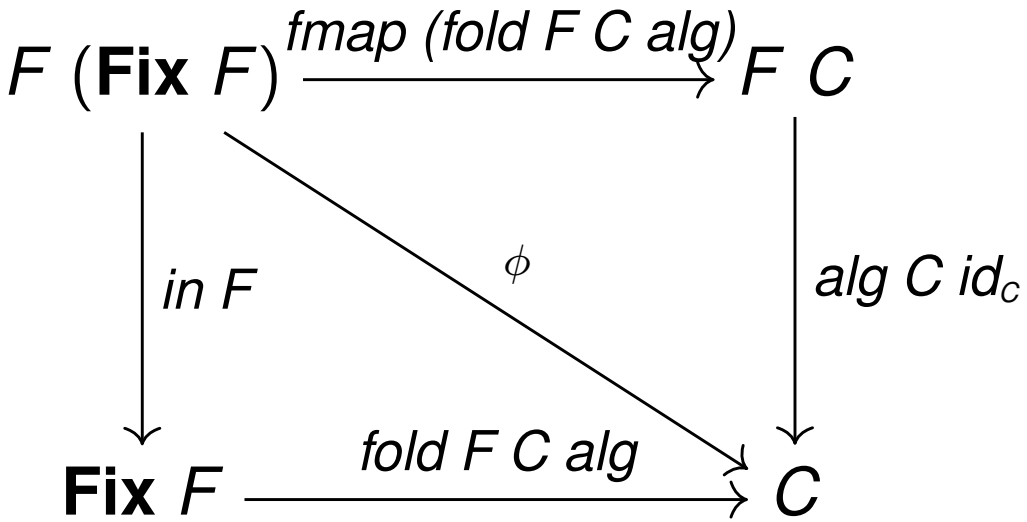
initial algebra semantics (conventional)



- need uniqueness of *fold*:

$$(h \circ in^{C} = alg \circ (fmap\ h)) \rightarrow (h = fold^{C}\ C\ alg)$$

initial algebra semantics (Mendler-style)



where $\phi := alg\ (\mathbf{Fix}\ F)\ (fold\ F\ C\ alg)$

- need commutativity of upper triangle
- need uniqueness of *fold*:

$$(h \circ in^M = alg\ (\mathbf{Fix^M}\ F)\ h) \rightarrow (h = fold^M\ C\ alg)$$

inductively defined relations as MDT

- consider unary relations (predicates) on type T
- $R : (T \to \mathrm{Prop}) \to T \to \mathrm{Prop}$
  endofunctor in diagram category $T \to \mathrm{Prop}$
- $P : T \to \mathrm{Prop}$ predicate on T
- T-indexed R-algebras on T-indexed carrier P

Church algebras and fixpoint:

- $\mathbf{Alg^{Cl}}\ T\ R\ P := \forall w : T.\ R\ P\ w \to P\ w$
- $\mathbf{Fix^{Cl}}\ T\ R\ w := \forall P.\ \mathbf{Alg^{Cl}}\ T\ R\ P \to P\ w$

Mendler algebras and fixpoint:

- $\mathbf{Alg^{Ml}}\ T\ R\ P :=$
  $\forall A.\ (\forall w : T.\ A\ w \to P\ w) \to \forall w : T.\ R\ A\ w \to P\ w$
- $\mathbf{Fix^{Ml}}\ T\ R\ w := \forall P.\ \mathbf{Alg^{Ml}}\ T\ R\ P \to P\ w$

example: inductive relations

conventional inductive predicate:

$$\begin{array}{l}\mathbf{IsTrm}: \ \mathbf{Trm} \to \mathbf{Prop} \quad =_{dt} \\ \quad \mathit{IsLit}\ (n: \mathbf{Nat}): \mathbf{IsTrm}\ (\mathit{Lit}\ n) \\ \quad \mathit{IsAdd}\ (e_1\ e_2: \mathbf{Trm}): \\ \qquad \mathbf{IsTrm}\ e_1 \to \mathbf{IsTrm}\ e_1 \to \mathbf{IsTrm}\ (\mathit{Add}\ e_1\ e_2)\end{array}$$

Trm-indexed functor:

$$\begin{array}{l}\mathbf{IsTrm_R}\ (P: \mathbf{Trm}): \ \mathbf{Trm} \to \mathbf{Prop} \quad =_{dt} \\ \quad \mathit{IsLit}\ (n: \mathbf{Nat}): \mathbf{IsTrm_R}\ P\ (\mathit{Lit}\ n) \\ \quad \mathit{IsAdd}\ (e_1\ e_2: \mathbf{Trm}): \\ \qquad P\ e_1 \to P\ e_1 \to \mathbf{IsTrm_R}\ P\ (\mathit{Add}\ e_1\ e_2)\end{array}$$

modular inductive predicate (Church-style):

$$\mathbf{IsTrm}: \ \mathbf{Trm} \to \mathbf{Prop} \quad := \quad \mathbf{Fix^{Cl}}\ \ \mathbf{Trm}\ \ \mathbf{IsTrm_R}$$

modular inductive predicate (Mendler-style):

$$\mathbf{IsTrm}: \ \mathbf{Trm} \to \mathbf{Prop} \quad := \quad \mathbf{Fix^{Ml}}\ \ \mathbf{Trm}\ \ \mathbf{IsTrm_R}$$

inductive proofs

- consider non-dependent induction (corresponding to iteration)
- for $T : \mathsf{Set}$ and $P : T \to \mathsf{Prop}$, find a proof

$$\Gamma, w : T \vdash \_? : X\ w \to P\ w \qquad (G)$$

  by induction on modular inductive type $X : T \to \mathsf{Prop}$
- problem: X is not syntactically a datatype,
  no induction principle supplied by Coq
- generic clue: fold a T-indexed algebra with carrier P
- however, choosing $X := \mathsf{Fix}^{\mathsf{Cl}}\ T\ R$, the algebra to fold is

$$\forall w : T.\ R\ T\ P\ w \to P\ w$$

  – hardly an induction step

Mendler-style induction (1)

Mendler-style induction: *induction hypothesis* given explicitly, *inductive call argument* typed with a fresh variable

$$\Gamma,\ A : \mathbf{Type},\ i\_hyp : \forall v : T.\ A\ v \to P\ v,\ w : T,\ i\_arg : R\ A\ w \ \vdash\ t : P\ w \qquad (1)$$

Coq *inversion* tactic applied to i_arg (to deconstruct R) can introduce inductive call arguments of type $A\ w$ in new subgoals

$$\Gamma,\ A : \mathbf{Type},\ i\_hyp : \forall v : T.\ A\ v \to P\ v,\ w : T, \ldots, i\_call\_arg_n : A\ w, \ldots \vdash s_t : P\ w$$

$\ldots$

freshness of A makes proof an iteration:
$i\_call\_arg_n$ only used in i_hyp, not further analysed

Mendler-style induction (2)

by abstracting (1) we get a Mendler algebra

$$\Gamma \vdash \lambda A\ i\_hyp\ w\ i\_arg.\ t : \quad (2)$$
$$\forall A.\ (\forall v : T.\ A\ v \to P\ v) \to \forall w : T.\ R\ A\ w \to P\ w$$

i.e. (2) can be rewritten

$$\Gamma \vdash \lambda A\ i\_hyp\ w\ i\_arg.\ t : \mathbf{Alg^{MI}}\ T\ R\ P \quad (3)$$

chosen $X := \mathrm{Fix}^{\mathrm{MI}}\ T\ R$, the original goal is obtained by folding (3)

$$\Gamma \vdash \mathit{fold}^{MI}\ T\ R\ P\ (\lambda A\ i\_hyp\ w\ i\_arg.\ t) : \quad (G)$$
$$\forall w : T.\ \mathbf{Fix^{MI}}\ T\ R\ w \to P\ w$$

general strategy and limitations

1) modularly defined, inductive relation over T, i.e. $\mathbf{Fix}^{\mathbf{MI}}\ T\ R$
2) property of interest P over T

---

build a modular proof as T-indexed Mendler algebra of the relation functor (i.e. R) with indexed carrier P, i.e. $\mathbf{Alg}^{\mathbf{MI}}\ T\ R\ P$

however – this works only with non-dependent induction

in general, inductive proofs involve dependence of the conclusion on the inductive argument

example: type preservation

type preservation

$$\textbf{TypPreseve}\ (e :\ \textbf{Trm}) :\ \textbf{Prop}\ :=$$
$$\forall t :\ \textbf{Typ}.\ \textbf{Typed}\ (e,\ t) \to \textbf{Typed}\ (\mathit{val2trm}\ (\mathit{eval}\ e),\ t)$$

conventionally provable by (dependent) induction on e

$$\Gamma, e : \textbf{Trm} \vdash\ \_? :\ \textbf{TypPreserve}\ e$$

example: predicatisation

predicatisation: prove by non-dependent induction on **IsTrm**

$$\Gamma, e : \mathbf{Trm} \vdash \_? : \mathbf{IsTrm}\ e \to \mathbf{TypPreserve}\ e$$

a modular inductive proof can then be obtained by constructing an indexed Mendler algebra

$$\mathbf{Alg}^{\mathbf{MI}}\ \mathbf{Trm}\ \mathbf{IsTrm}_{\mathbf{R}}\ \mathbf{TypPreserve}$$

residual goal to be proved by dependent induction

$$\Gamma, e : \mathbf{Trm} \vdash \_? : \mathbf{IsTrm}\ e$$

MTC-style induction

- MTC provides generalised induction for MDTs
- relies on the universal property of *fold*
  - existence and uniqueness of *fold* = initiality of the fixpoint
  - very strong property, guaranteeing semantic soundness
- inductionless induction, based on algebraic reasoning
  - proof algebras defined with Σ types,
    a principle we call Σ *induction*

Σ induction

premises:

(1) universal property for functor F

(2) existence of a well-formed algebra **Alg** F $(\Sigma x : T.\ P\ x)$

conclusion:

$\forall x : T.\ P\ x$

well-formedness is a condition on terms (algebras), it can be turned into one on types (functors) by introducing a *weak induction principle* (as a filtering condition on terms)

example: discharging predicatisation totality

predicatisation totality hypothesis: totality of the predicate over the datatype it represents

$$\forall w : \mathbf{Trm}.\ \mathbf{IsTrm}\ w$$

discharged by MTC-style induction

(1) define well-formed proof algebra

$$\begin{aligned}
&\mathbf{isExpPrfAlg}\ (t :\ \mathbf{Exp_F}\ (\Sigma x.\ \mathbf{IsExp}\ x)) : \\
&\qquad\qquad \Sigma x.\ \mathbf{IsExp}\ x\ :=\ \text{match } t \text{ with} \\
&\quad \mid \mathit{Lit}\ n\ \Rightarrow\ \text{exist}\ (\mathit{in}^{MI}\ (\mathit{IsLit}\ _\ n)) \\
&\quad \mid \mathit{Add}\ e_1\ e_2\ \Rightarrow\ \text{exist}\ (\mathit{in}^{MI}\ (\mathit{IsAdd}\ _\ _\ _ \\
&\qquad\qquad (\text{proj2_sig}\ e_1)\ (\text{proj2_sig}\ e_2)))
\end{aligned}$$

(2) apply Σ induction, assuming the universal property

predicatisation approach

total predicatisation of a functor F: an indexed functor R on $\mathbf{Fix}^{\mathbf{M}}\ F$, such that

$$\forall w : \mathbf{Fix}^{\mathbf{M}}\ F.\ \mathbf{Fix}^{\mathbf{MI}}\ (\mathbf{Fix}^{\mathbf{M}}\ F)\ R$$

using predicatisation: single application of MTC-style induction for each MDT, to discharge totality

question 1: initiality

Mendler-style induction: not strictly dependent on universal property of *fold*

however – how to guarantee the model satisfies it?

currently in our development: undischarged premise

MTC approach: packing the universal property with fixpoint objects, using Σ types

other approaches?

question 2: predicatisation

generate the isomorphic predicate (unique up to isomorphism)

- given functor F, generate the indexed functor R that has the *-same-* structure as F
- e.g. in our example the *fold* of **IsExpPrfAlg** is invertible

discharge totality of the predicate

- consider cases when structural induction (correspondingly, MTC induction) does not suffice

conclusion

- modularity essential for reuse and cost-effective verification
- Mendler induction can be used to simplify reasoning about modular datatypes, increasing type-directedness and reducing boilerplate
- well-suited to language semantics in SOS (case study in [FICS'15])

  `http://cs.swan.ac.uk/ cspt/MDTC`
- generality achieved by integration with MTC-style induction (current work)

  `https://bitbucket.org/ptorrx/modind`

conclusion

- modularity essential for reuse and cost-effective verification
- Mendler induction can be used to simplify reasoning about modular datatypes, increasing type-directedness and reducing boilerplate
- well-suited to language semantics in SOS (case study in [FICS'15])

  `http://cs.swan.ac.uk/ cspt/MDTC`
- generality achieved by integration with MTC-style induction (current work)

  `https://bitbucket.org/ptorrx/modind`

Thanks for your attention!