Substitution systems revisited

Benedikt Ahrens [1] Ralph Matthes[2]

[1]Institut de Recherche en Informatique de Toulouse, Université Paul Sabatier
[2]CNRS and IRIT, Université Paul Sabatier

Types 2015 in Tallinn, Estonia
Talk on May 21, 2015
Handout version of May 22—no revelation in stages
minor revision on June 4

Outline

1. What is substitution?
2. What are substitution systems?
3. Some new theoretical results
4. Formalization in univalent mathematics

Outline

Parallel substitution in a simple framework

Given a first-order signature over some supply of variables, substitution is a homomorphism: the substitution function commutes with all term-forming operations

Notation:

- TA for the set of terms over variable supply A (potentially free in the terms)
- $f : A \to TB$ is called a substitution rule
- $[f] : TA \to TB$ is the substitution function for rule f, written post-fix

For $t : TA$, the result of the parallel substitution according to f is thus written as $t[f]$ and belongs to TB.

Substitution: a problem?

Parallel substitution in λ-calculus: same idea, looks up f at variables, should just commute with all term constructors

- $x[f] = fx$
- $(MN)[f] = M[f]N[f]$
- $(\lambda x M)[f] = \lambda x(M[f])$

Of course, this has to be done capture-free: x not free in any fy substituted for a free variable y of M.

Either one restricts the allowed substitution rules f or one builds α-equivalence into the system.

We are not interested in such low-level details and want to work algebraically / categorically.

Typed de Bruijn indices

De Bruijn indices solve the problem but are too untyped: what corresponds to the variable supply A in the notation TA?

Rather a typeful version of de Bruijn indices with nested datatypes (Altenkirch & Reus 1999, Bird & Paterson 1999): solve the equation

$$TA = A + TA \times TA + T(\mathit{option}\, A)$$

$\mathit{option}\, A$ is A and one extra element for the name of the variable that may be bound in the λ case—this is locally nameless.

Reference to a more complicated argument in one of the cases is the distinctive feature of nested datatypes.

In 2004, it was quite difficult to define T as an inductive family in Coq and to program $t[f]$ by recursion over t. Nested datatypes have been explicitly supported in Coq shortly afterwards (thanks to Christine Paulin).

We get a monad

Inclusion of variables $\mathit{var} : A \to TA$ together with the operation $\cdot[\cdot]$ yield the signature of a monad with unit $\eta := \mathit{var}$ and the monadic "bind" operation given by substitution typed as

$$\cdot[\cdot] : (A \to TB) \to TA \to TB$$

The monad laws can be shown to hold.

- a law saying that substitution on variables is look-up
- substituting the term $\mathit{var}\, a$ for every variable a does not change a term
- a law about the effect of two subsequent substitutions

What do the monad laws have to do with our initial goal of defining substitution as a homomorphism? Application and λ-abstraction are not mentioned in the monad laws.

Monad multiplication

Use substitution with $A := TB$ and $f := \lambda x^{TB}.x$.
$\mu := [f] : T(TB) \to TB$ does what?

In a term whose variables have as names terms over B, replace those names by themselves, but now seen as terms that are "integrated" into the result term.

You have seen this operation in the tutorial by Joachim Kock on Tuesday afternoon. In other words, monad multiplication μ removes the cross section between the trunk of the term and the term-like variable leaves.

Substitution is expressed in terms of μ and renaming

$$[f] = \mu \circ Tf,$$

where, in general, for $f : A \to B$, we write $Tf : TA \to TB$ for the renaming of variables names according to f.

Pointwise: $t[f] = \mu(T\, f\, t)$.

Explicit substitution, explicit flattening

Explicit substitution does not denote the result of carrying out a substitution but the task of doing the substitution. There is an extra constructor for that, like the well-known `let x := M in N` that binds `x` in `N`.

Forms of explicit substitution:

- delayed substitution: the constructor can be replaced by the result of the intended operation in one step
- small-step semantics: the explicit substitution can be distributed over the term constructors and is executed at the leaves or by way of "garbage collection"; there can be rules that describe the interchange of explicit substitutions.

Analogously, we want an explicit monad multiplication, a constructor of the same type as μ that may be evaluated later into a "true" monad multiplication (in the sense of delayed substitution). It "flattens" $T(TB)$ into TB only formally.

Explicit flattening—a formal μ

We solve the extended equation in T'

$$T'A = A + T'A \times T'A + T'(\mathit{option}\, A) + T'(T'A)$$

The constructor for the last summand of type $T'(T'A) \to T'A$ is the formal / explicit flattening operation. All explicit flattening can be evaluated by a function $\mathit{eval} : T'A \to TA$ (T for the usual λ-terms; an example we carried out in our setting for this talk).

T' cannot be defined as inductive family in Coq. In Agda, this is possible when assuming Type:Type (Robin Adams tried this out on Tuesday). Renaming with f, written $T'f$ can be defined in Agda, but not be checked to terminate although this uses only plain iteration (see TCS paper 2005 by Abel, Matthes and Uustalu).

Disclaimer

We do not insist on having explicit flattening in the syntax of lambda calculi. It is an example of a natural construction that does not only bind a fixed finite number of variables.
It is perfectly captured by the monadic take on substitution and also by the heterogeneous substitution systems, recalled and further developed in this talk.
Evaluation of explicit flattening is an example of a morphism of heterogeneous substitution systems, for the first time described in this talk.

Outline

Substitution should be a homomorphism

Tarmo Uustalu and the second author identified a general framework for getting well-behaved substitution in their 2004 TCS paper.

The equations for T and T' in the introduction were both of the shape $TA = A + HTA$. Without points, this is $T = \mathsf{Id} + HT$. From right to left, the solution consists of variable inclusion $\eta : \mathsf{Id} \to T$ and an H-algebra $\tau : HT \to T$.

We think of this data properly in categorical terms: Let $\mathcal{C}$ be a category with finite coproducts (one may think of $\mathtt{Set}$), T an endofunctor on $\mathcal{C}$ and H an endofunctor on $[\mathcal{C}, \mathcal{C}]$.

Monad multiplication $\mu : T \cdot T \to T$ is then required to fulfill $\mu \circ \eta \cdot T = \mathsf{id}_T$ (the first monad law) and

$$\mu \circ \tau \cdot T = \tau \circ H\mu \circ \theta$$

with a given natural transformation $\theta : HT \cdot T \to H(T \cdot T)$.

What is θ?

In the 2004 TCS paper, θ is part of a "strength-like datum" that can be constructed for each term constructor individually, corresponding to H being a sum / coproduct of H_i.

Case $HTA = TA \times TA$ assuming finite products in $\mathcal{C}$: The type of θ applied to A is
$(HT \cdot T)A \to H(T \cdot T)A = T(TA) \times T(TA) \to T(TA) \times T(TA)$,
and θ can be set to the identity. This corresponds to the triviality of first-order operations in substitution.

Case $HTA = T(\mathit{option}\, A)$, i. e., $HT = T \cdot \mathit{option}$.
$HT \cdot T \to H(T \cdot T) = T \cdot \mathit{option} \cdot T \to T \cdot T \cdot \mathit{option}$. It suffices to get from $\mathit{option}(TA)$ to $T(\mathit{option}\, A)$. This is the usual lifting operation needed for substitution on λ-abstraction.

Case $HTA = T(TA)$ for explicit flattening, i. e., $HT = T \cdot T$. Requires θ from T^3 to T^4 with four possibilities of extending $\eta : \mathsf{Id} \to T$ to this type. The "right one" is $\theta = T \cdot \eta \cdot T^2$.

The truth about θ

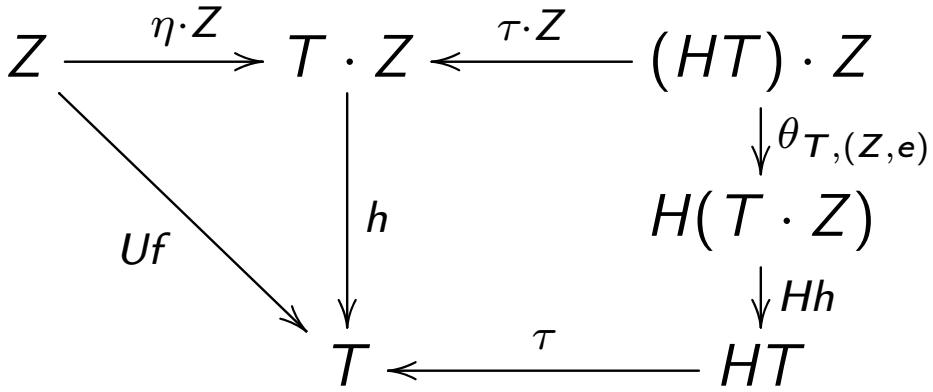
θ of the previous slide is just the family member $\theta_{T,(T,\eta)}$ of a natural transformation $\theta : (H-) \cdot U\sim \to H(- \cdot U\sim)$ between functors $[\mathcal{C},\mathcal{C}] \times \mathbf{Ptd}(\mathcal{C}) \to [\mathcal{C},\mathcal{C}]$ with $\mathbf{Ptd}(\mathcal{C})$ the category of pointed functors, whose objects are endofunctors Z of $\mathcal{C}$ together with a "point" $e : \mathsf{Id} \to Z$. The most typical example is just (T,η), as used for the instance on the previous slide. The functor U "forgets" the point.
θ has to be compatible with the monoid structure in its second argument (giving rise to two equational laws).

In our example of $HT = T^2$, the type of $\theta_{X,(Z,e)}$ is $X \cdot X \cdot Z \to X \cdot Z \cdot X \cdot Z$ which suggests the definition $\theta_{X,(Z,e)} := X \cdot e \cdot X \cdot Z$ and excludes all other possibilities of the previous slide that were possible for the special case $\theta_{T,(T,\eta)}$.

The other examples can be extended to this parameterized situation without problems, see p. 165 of the 2004 TCS paper.

Definition of substitution system

Given the parameters H and θ, the data T, η and τ forms a heterogeneous substitution system w. r. t. H and θ (an $\mathsf{hss}(H, \theta)$) iff for every $\mathbf{Ptd}(\mathcal{C})$-morphism $f : (Z, e) \to (T, \eta)$, there exists a unique $[\mathcal{C}, \mathcal{C}]$-morphism $h : T \cdot Z \to T$, denoted $\{f\}$, satisfying



In equations:

- $h \circ \eta \cdot Z = Uf$
- $h \circ \tau \cdot Z = \tau \circ Hh \circ \theta_{T,(Z,e)}$

f is a $\mathbf{Ptd}(\mathcal{C})$-morphism means that its underlying $\mathcal{C}$-morphism Uf satisfies $Uf \circ e = \eta$ (Uf compatible with the points e and η).

Getting back monad multiplication

The identity on T is compatible with η, hence yields a $\mathbf{Ptd}(\mathcal{C})$-morphism $\mathsf{id}_{(T,\eta)}$ from (T,η) to itself. Then, in an hss,

$$\mu := \{\mathsf{id}_{(T,\eta)}\} : T \cdot T \to T$$

fulfills the equations we asked for in the first place.

From the uniqueness of the substitution operation, we can deduce that

$$\{\} : \mathbf{Ptd}(-,(T,\eta)) \to [\mathcal{C},\mathcal{C}](T \cdot U-, T)$$

is a natural transformation. Its most prominent instance is:

$$\{f\} = \mu \circ T \cdot Uf : T \cdot Z \to T$$

Compare this with the usual situation in a monad recalled before:

$$[f] = \mu \circ Tf : TA \to TB$$

Main result of TCS 2004 paper

The notion of hss does not require T to be the support of an initial algebra! TA does not necessarily denote the wellfounded terms over the signature described by H. Like in Haskell, it could also be the greatest fixed-point, including the non-wellfounded expressions. The notion of hss is not limited to any of these two cases and must not be limited to them, in order to accommodate our example of evaluation of explicit flattenings.

Main theorem of 2004 TCS paper: every hss gives rise to a monad via $\mu := \{\mathsf{id}_{(T,\eta)}\}$.

The proof very much exploits the unicity of the substitution operation $\{f\}$ for two different choices of f (with different types).

About initial and final solutions

Further main results of TCS 2004 paper (sketchy presentation):

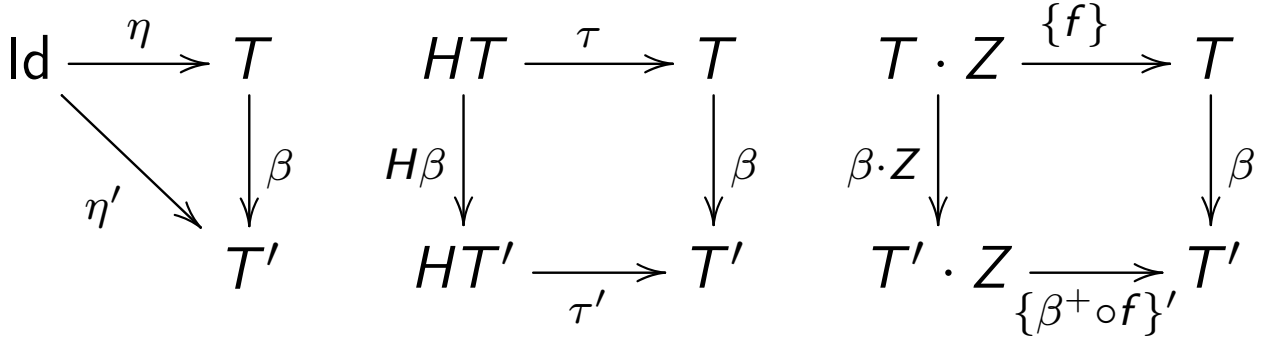
- If there is an initial $(\mathsf{Id} + H-)$-algebra and (technical condition) one can form right Kan extensions for $[\mathcal{C}, \mathcal{C}]$-elements, then the structure map of this algebra gives rise to an hss.
- If there is a final $(\mathsf{Id} + H-)$-coalgebra, then the inverse of its structure map gives rise to an hss.

In the initial case, this operation uses generalized iteration of Bird and Paterson. One may argue that the defining equations for $\{f\}$ are already the definition: the operation is determined pointwise, but the intricate recursive call pattern needs justification—as provided by the scheme of Bird and Paterson. In the final case, proper corecursion is needed, but a Haskell programmer might still argue that the defining equations for $\{f\}$ *are* the definition. But, of course, we want to be sure that the algorithm is productive.

Outline

Organizing the hss into a category

The following natural question was left out from the TCS 2004 paper: do the heterogeneous substitution systems form the objects of a category of interest?
The data for an $\mathsf{hss}(H, \theta)$ contains a pointed morphism (T, η) and an H-algebra τ. So, a morphism from $(T, \eta, \tau, \{\})$ to $(T', \eta', \tau', \{\}')$ ought to be a natural transformation $\beta : T \to T'$ that respects the points and the algebra structures and additionally "commutes" with the substitution operations. Technically, this means that the following diagrams commute:



(β^+ is β seen as a $\mathbf{Ptd}(\mathcal{C})$-morphism thanks to the first rule.)

A functor from hss into monads

The map from heterogeneous substitution systems to monads given by the main theorem of the 2004 paper is the object map of a functor $\mathsf{hss}(H, \theta) \to \mathsf{Mon}(\mathcal{C})$.

The morphism map does not transform its argument β. One only verifies that it is even a monad morphism. The proof is easy by instantiating f to $\mathsf{id}_{(T,\eta)}$ as before and by using $\{\beta^+\} = \mu' \circ T' \cdot \beta$ (which follows from naturality of $\{\}'$ as before).

Of course, the monads do not express the algebraic structure, so our functor into monads may forget data and will thus not be full. But it is faithful.

We are now leaving the implemented part of the new results.

Initiality in the category of hss

Recall the "old" theorem: If there is an initial $(\mathsf{Id} + H-)$-algebra and one can form right Kan extensions for $[\mathcal{C}, \mathcal{C}]$-elements, then the structure map of this algebra gives rise to an hss.

Under the same conditions, we can even show that we obtain an initial object in the category $\mathsf{hss}(H, \theta)$. To show this, we used the general fusion law of Bird and Paterson 1999 that is justified through right Kan extensions.

As an aside: syntactic readings of those results of Bird and Paterson were crucial for the analysis of a variety of (co)iteration schemes on nested datatypes in the TCS 2004 paper of Abel, Matthes and Uustalu. The non-functional properties (termination) were there, but no computer support for the verification of functional properties, which are provided through the results reported here.

Case study: evaluation of explicit flattening

Recall the λ-terms with explicit flattening given by the equation

$$T'A = A + T'A \times T'A + T'(\mathit{option}\, A) + T'(T'A)$$

We want to evaluate these terms by a morphism $\mathit{eval} : T' \to T$ that should come from initiality. We therefore have to equip the smaller syntactic domain T with a substitution operation concerning the richer signature corresponding to T'. In particular, as part of the hss object, we have to give an algebra structure for the richer signature and then prove properties dictated by hss. This heterogeneous substitution system does not stem from an initial algebra nor a final coalgebra. By the initiality theorem, there is then a unique hss morphism from the canonical hss for T' to this specific hss for T, our eval operation.

Outline

About UniMath

- Library of "univalent mathematics"
- Based on the Coq proof assistant
- Restriction to MLTT, the core of CIC, in particular no use of general inductive types, no records
- No HITs—propositional truncation implemented via impredicativity
- Lack of resizing rules in Coq is remedied by $\mathsf{Type} : \mathsf{Type}$

Why do we use UniMath?

- Extensional features of UF compared to IMLTT are crucial for avoiding setoids in implementation of categories
- Univalent Foundations is a principled way of adding those extensional features axiomatically
- Reusing and extending the existing library of category theory

Challenges when formalizing results in UF

- Passing from a "very extensional" setting to a not quite as extensional one

$\rightsquigarrow$ mathematically well-typed statements become non-welltyped (w. r. t. the unaltered Coq type-checker without universe constraints)

- More specifically, main problem is source and target of natural transformations such as:

$$\alpha : F \cdot \mathsf{Id} \to G \quad \text{vs.} \quad \alpha : F \to G$$

- HoTT teaches us how to reason about transport, but it is still best **not** to transport

A well-hidden problem

- Functor law $F(f \circ g) = Ff \circ Fg$, for $f : a \to b$ has many occurrences of (implicit) parameter a.
- Can have non-convertible types $a \not\equiv a'$ such that $F_{a,c}(f \circ_{a',b,c} g)$ is still well-typed, e.g.,

$$f : \quad \Big(F \cdot (F \cdot F) \to F\Big) \quad \equiv \quad \Big((F \cdot F) \cdot F \to F\Big)$$

- Rewriting with functor law fails mysteriously in such a situation...
- After applying H to f, we lose convertibility of the types:

$$Hf : \quad \Big(H(F \cdot (F \cdot F)) \to HF\Big) \quad \not\equiv \quad \Big(H((F \cdot F) \cdot F) \to HF\Big)$$

Overview of the formalization

- Main theorem of Matthes & Uustalu 2004 formalized
- (Pre)category of hss, and proof that functor to monads is faithful
- approx. 2000 loc
- main difficulty in the proofs: Coq `rewrite` tactic unable to find the subterm to replace, even when hypothesis is fully instantiated

Conclusion I

A very general notion of terms involving binding has been studied. The concept of substitution systems embodies the vision that substitution is essentially a homomorphism. Monad laws are not built into the systems but derived. The notion is not dependent on representation of a least or greatest fixed point.

As original contribution, we organized heterogeneous substitution systems into a category and found that the structure of the category of monads is implied. We constructed initial hss from initial algebras and extended the example of evaluation of explicit flattening to this richer categorical setting.

Conclusion II

The notions and basic properties, as well as the result that monads and monad morphisms are obtained, are formally represented by help of the UniMath library in the Coq system. Extensional reasoning was mostly possible, but fine-tuning was necessary for formulations of properties in order to have them type-check since the ambient system Coq is intensional type theory.

The system is in place so that concrete algorithms on these (generalized) term structures could be mechanically verified for their functional properties.

Advertisement for FICS 2015

FICS 2015 - 10th International Workshop on Fixed Points in Computer Science
11 and 12 September 2015, Berlin, Germany
Affiliated to CSL 2015

- June 16, 2015: abstract submission
- June 23, 2015: paper submission

Typical submissions would be 8 pages long but submissions in the range $[6, 15]$ pages will be considered acceptable.

Invited speakers: Bartek Klin (Warsaw University), James Worrell (University of Oxford)

Program Chairs: Matteo Mio and Ralph Matthes

`http://www.irit.fr/FICS2015/`