A specification for Agda Core's unification algorithm for generic pattern matching

Ewen Broudin-Caradec, supervised by Jesper Cockx

TU Delft

école normale supérieure paris–saclay

Extended abstract

Generic pattern matching

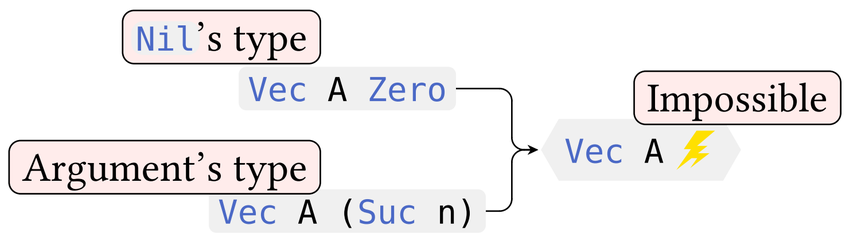
In Agda and others proof asssistants, partial functions are not accepted. However, we can still write the head function for non-empty vectors as follows:

```
data Vec (A : Set ) : ℕ → Set where
Nil : Vec A Zero
Cons : (n : ℕ) → A → Vec A n → Vec A (Suc n)
```

```
head : {A : Set} {n : ℕ} → Vec A (Suc n) → A
head (Cons _ a _) = a
```

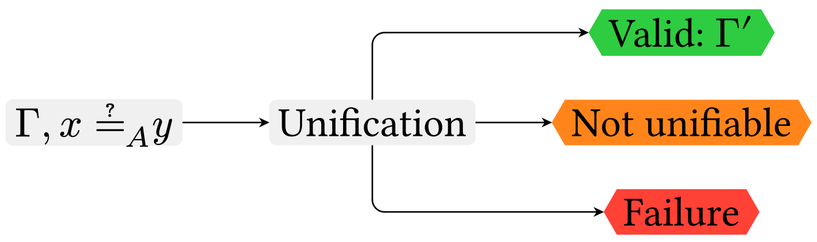
Where is the `Nil` case ?

The `Nil` case can safely be omitted. Indeed, the type that this pattern would have (`Vec A Zero`) is incompatible with the type that we already know the term has (`Vec A (Suc n)`). This incompatibility is found via unification.



Unification rules

Unification is used to solve telescopic equalities by checking that induced constraints are consitent and by instanciating variables.



It relies on a set of rules [1], positive for sucessful steps and negative for impossible cases.

Positive	Negative
Solution	Conflict
Deletion	Cycle
Injectivity	

$$\frac{x \in \text{variables} \quad t \text{ independent of } x}{\Gamma \vdash x \stackrel{?}{=}_A t \to \Gamma[x := t]}$$

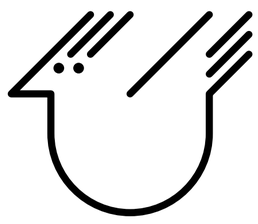
Simplified example of a rule: Solution

Unification and Axioms

Unification rules can hide various axioms like Uniqueness of Identity Proof (UIP) or Injectivity of Type Constructors [1]. In a general purpose proof assistant like Agda, we want to minimise the number of axioms as they limit the theories that one can implement. For example, UIP is incompatible with with homotopy type theory.

Agda Core

Agda Core is a project of core language for Agda [2]. Increasing trust in proofs is the very motivation of proof assistants, so it is crucial to have guarantees that their code is correct. Agda Core does it by writing the most important fragments of Agda in Agda to formalise and check them. It can also recheck proofs validated by Agda.



A small, formalised and trusted code base for Agda.

Unification example in 5 steps

Let $\Gamma := \mathtt{a} : \mathtt{A}, \mathtt{n} : \mathbb{N}, \mathtt{b} : \mathtt{A}, \mathtt{v} : \mathtt{Vec\ A\ n}$,

$$(1) \qquad (\Gamma), (e : \mathtt{0} \stackrel{?}{=}_{\mathbb{N}} \mathtt{n})(e_v : \mathtt{Cons\ 0\ a\ Nil} \stackrel{?}{=}_{\mathtt{Vec\ A\ (Suc\ e)}} \mathtt{Cons\ n\ b\ v})$$

Let $\Gamma_2 := \mathtt{a} : \mathtt{A}, \mathtt{b} : \mathtt{A}, \mathtt{v} : \mathtt{Vec\ A\ 0}$,

$$(2) \qquad (\Gamma_2), (e_v : \mathtt{Cons\ 0\ a\ Nil} \stackrel{?}{=}_{\mathtt{Vec\ A\ 1}} \mathtt{Cons\ 0\ b\ v})$$

$$(3) \qquad (\Gamma_2), (e_0 : \mathtt{0} \stackrel{?}{=}_{\mathbb{N}} \mathtt{0})(e_1 : \mathtt{a} \stackrel{?}{=}_{\mathtt{A}} \mathtt{b})(e_2 : \mathtt{Nil} \stackrel{?}{=}_{\mathtt{Vec\ A\ 0}} \mathtt{v})$$

$$(4) \qquad (\Gamma_2), (e_1 : \mathtt{a} \stackrel{?}{=}_{\mathtt{A}} \mathtt{b})(e_2 : \mathtt{Nil} \stackrel{?}{=}_{\mathtt{Vec\ A\ 0}} \mathtt{v})$$

Let $\Gamma_5 := \mathtt{a} : \mathtt{A}, \mathtt{v} : \mathtt{Vec\ A\ 0}$,

$$(5) \qquad (\Gamma_5), (e_2 : \mathtt{Nil} \stackrel{?}{=}_{\mathtt{Vec\ A\ 0}} \mathtt{v})$$

$$(6) \qquad (a : A), ()$$

Contribution

I implemented data structures for telescopic equality and specified the unification rules in Agda, within the Agda Core repository.
I also wrote an algorithm to check if variables can be swapped in a context.

Future work

The next step is to use this specification in the implementation of a unification algorithm for Agda Core. Once this algorithm is programmed, further work will be needed on the existing type checking algorithm to call the unification algorithm when it is needed.

Bibliography

[1] J. Cockx and D. Devriese, "Proof-relevant unification: Dependent pattern matching with only the axioms of your type theory."

[2] B. Liesnikov and J. Cockx, "Building a Correct-by-Construction Type Checker for a Dependently Typed Core Language,"

GitHub repository of Agda Core →