Using reflection to solve some differential equations

Guillaume Allais
Junior Laboratory COQTAIL
ENS Lyon - France

June 28, 2011

Abstract

On top of COQTAIL's libraries that provide a formalization of power series, we added a small development that aims at simplifying proofs that given power series are solutions of specific differential equations. The use of reflection allows to prove general facts about differential equations which can then be used to simplify the proofs thanks to an `Ltac` machinery that performs the tedious conversions.

Files: All the implementation mentioned in this paper are available for download via COQTAIL's svn repository[1].

1 Definitions

Even if the `Dequa` library relies heavily on COQTAIL's `Rpser` definitions and developments, one does not need to know much about these in order to read this presentation. We recall here the basic definitions that are used afterwards.

The sequence of coefficients of the power series defining the constant function $\lambda\_.C$ is given by the `An_cst` function. The infiniteness of its radius of convergence is obviously part of the library.

$$\texttt{An\_cst}\,(C) := n \mapsto \begin{cases} C & \text{if } n = 0 \\ 0 & \text{otherwise} \end{cases}$$

The coefficients of the formal k^{th} derivative of a power series defined by A_n is given by the `An_nth_deriv` function. The link between the formal derivative and the actual one is also stated and proved in the available libraries.

$$\texttt{An\_nth\_deriv}\,(A_n, k) := n \mapsto \frac{(n+k)!}{n!} A_{n+k}$$

Given a sequence of coefficients A_n and a proof ρ that the convergence radius of $\sum_n A_n x^n$ is infinite, the sum of the power series is given by the function `sum`:

$$\texttt{sum}\,(A_n, \rho) := x \mapsto \sum_{k=0}^{+\infty} A_k x^k$$

2 Reflection of differential equations

In order to represent the differential equations, we need two components: a datatype which can describe the structure of a differential equation and a semantics which, given a structure and a context, yields a formula in `Prop` stating that the functions in the context are solutions of the described differential equation.

[1] see `http://sourceforge.net/projects/coqtail/develop` and in particular `src/Reals/Dequa_*.v`.

2.1 Reification

A differential equation $\forall x.e1(x) = e2(x)$ is represented as a pair (E_1, E_2) of side equations (usually written $E_1 :=: E_2$) where E_1 (resp. E_2) represents e_1 (resp. e_2). A side equation is either a constant, a function's n^{th} derivative or a linear combination of side equations. We use the following datatype to describe such side equations:

```
Inductive side_equa : Set :=
    | cst  : forall (r : R), side_equa
    | scal : forall (r : R) (s : side_equa), side_equa
    | y    : forall (p : nat) (k : nat), side_equa
    | opp  : forall (s1 : side_equa), side_equa
    | plus : forall (s1 s2 : side_equa), side_equa.
```

This representation can be rather easily extended with new data constructors when the corresponding theory has been formalized in Coq [1]: adding the multiplication by a scalar (which was not present in our first toy example) was a matter of less than twenty minutes as soon as all the appropriate lemmas were available in `Rpser`[2].

2.2 Interpretation

Unlike other procedures using reflection, `Dequa` provides various semantics for these side equations: the straightforward one (à la Tarski) but also a semantics that yields equations on sequences of coefficients over $\mathbb{R}$.

These semantics ($\mathbb{K}$ is either $\mathbb{R}$ or $\mathbb{N}$) are defined in two steps: given an environment, $\texttt{interp}_{\mathbb{K}}$ translates side equations and $[|\_|]_{\mathbb{K}}$ uses it to interpret equations. For the sake of simplicity, instead of enforcing the well-formedness of the environment by a dependent type (e.g. a dependent vector), the defined functions are partial and use the `option` monad.

`Rseq_infty` is the subset of sequences over $\mathbb{R}$ such that the corresponding power series has an infinite convergence radius (a dependent pair in Coq).

2.2.1 Equations on power series

Our first concern is to define the semantics à la Tarski that translates `diff_equa` as differential equations on sums of power series. $\texttt{interp}_{\mathbb{R}}$: `side_equa` $\rightarrow$ `list Rseq_infty` $\rightarrow$ `option (R` $\rightarrow$ `R)` translates constants as the constant function, variables as derivatives of sums of power series of the corresponding `Rseq` available in the environment and linear combinations as linear combinations of functions[3].

$[|\_|]_{\mathbb{R}}$: `diff_equa` $\rightarrow$ `list Rseq_infty` $\rightarrow$ `Prop` is the corresponding interpretation function: given the interpretation of two side equations, it states that the obtained functions are extensionally equal.

2.2.2 Equations on sequences over $\mathbb{R}$

The second semantics translates `diff_equa` as equations on sequences over $\mathbb{R}$. The function $\texttt{interp}_{\mathbb{N}}$: `side_equa` $\rightarrow$ `list Rseq` $\rightarrow$ `option Rseq` translates constants as `An_cst`, variables as `An_nth_deriv` and linear combinations as linear combinations of sequences[4].

$[|\_|]_{\mathbb{N}}$: `diff_equa` $\rightarrow$ `list Rseq` $\rightarrow$ `Prop` is the corresponding interpretation function: given the interpretation of two side equations, it states that the obtained sequences are extensionally equal.

[2] We hope to modify the data structure to be able to talk about the product of two functions in a near future.
[3] See 3 for technical details.
[4] See 3 for technical details.

2.3 Relation between these semantics

The relation between these semantics comes from two simple facts: given two sequences of coefficients extensionally equal, the corresponding power series are equal and sums of power series are compatible with sum (the sum of the power series is the power series of the sum), opposite, etc. Therefore it is sufficient to prove that some coefficients are solutions of the equation on sequences in order to prove that the sums of the corresponding power series are solutions of the differential equations.

We define $proj_1$ the projection which, given a sequence A_n of type `Rseq_infty`, forgets the proof that $\sum_n A_n x^n$ has an infinite convergence radius: $proj_1 A_n$ is of type `Rseq` (in Coq we simply use `projT1`).

Theorem 1 *Our main result states that given a context ρ of type* `list Rseq_infty`*:*

$$[|e_1 :=: e_2|]_{\mathbb{N}}\,(map\ proj_1\ \rho) \Rightarrow [|e_1 :=: e_2|]_{\mathbb{R}}\,\rho$$

ie. we can prove that some functions (sums of power series) are solutions of a given differential equation by proving results on their coefficients.

Proof The proof uses the following intermediate lemma:

$$\texttt{interp}_{\mathbb{N}}(e, map\ proj_1\ \rho) = \texttt{Some}\,U_n \wedge \texttt{interp}_{\mathbb{R}}(e,\rho) = \texttt{Some}\,f$$

$$\Rightarrow \exists\rho.\forall x.f(x) = \texttt{sum}\,(U_n,\rho)(x)$$

which is proved by induction on e.

2.4 Going back to actual problems

We just proved a theorem that gives us the opportunity to show that some functions are solutions of differential equations just by looking at the coefficients of their power series. This theorem is however quite useless without tactics on top of it: one has to feed it with a `diff_equa` and an environment and it will output a statement that has a very specific shape (given by the function $[|\_|]_{\mathbb{R}}$) which is highly unlikely to match the current goal.

Because quoting differential equations and normalizing goals is tedious enough for us to want to avoid doing it by hand, we wrote a couple of tactics that do all the hard work for us. We have two main kinds of tactics corresponding to the two main steps: quoting and normalizing.

2.4.1 Quoting differential equations

There are different ways of quoting the same differential equation depending on whether you care about the proofs (that the radius of convergence is infinite, that you can take the derivative of the power series, etc.) used or not and whether a constant has to be a value or not.

We chose not to care about the proofs because none of the values computed by the operations we consider are influenced by the proof used. Therefore $\forall x.\,\texttt{sum}(a_n, pr_1)(x) = \texttt{sum}(a_n, pr_2)(x)$ will be represented as $\texttt{y}(0,0) = \texttt{y}(0,0)$.

We also chose to allow constants to be complex expressions and not only values in order to guarantee that the generated `diff_equa` will be as simple as possible. We will therefore quote $\forall x.\,\texttt{sum}(a_n, pr_1)(x) = 5*(4+3)$ as $\texttt{y}(0,0) = \texttt{cst}(5*(4+3))$ rather than $\texttt{y}(0,0) = \texttt{scal}(5, (\texttt{plus}((\texttt{cst}(4)), (\texttt{cst}(3)))))$.

Given this choices, the tactic machinery should look like:

- $\texttt{isconst}(s, x)$ that checks whether the expression s is a constant with respect to x.
- $\texttt{add\_variables}(\texttt{a}_\texttt{n}, \texttt{pr}, \texttt{env})$ that adds (a_n, pr) to the environment *env* if there is no (a_n, pr') already declared[5] and returns an updated environment together with an integer representing (a_n, pr).

[5] where pr, pr' are proofs that a_n has an infinite convergence radius.

- `quote_side_equa`$(\texttt{env}, \texttt{s}, \texttt{x})$ that proceeds as follow:

  if `isconst`(s, x) then return `cst`(s) and *env*

  else if the head constructor of s is in $\{-\_; \_ + \_; \_ - \_\}$ then translate it to the appropriate `side_equa` constructor (either `opp`, `plus` or `minus`) and recursively quote the subexpressions while propagating the environment

  else if the head constructor of s is $\_ * \_$ then one of the subexpressions has to be a constant[6]; recursively quote the non-constant equation and output a `scal` together with the updated environment

  else if the expression is a sum or a derivative, add the corresponding sequence to the environment *via* `add_variables` that returns a $p \in \mathbb{N}$ and an updated environment and propagate the environment while outputting `y`(p, k) where k is such that the quoted expression is a k^{th} derivative.

2.4.2 Normalizing goals

The aim of the goal's normalization procedure is to modify the goal so that it matches $[|E|]_{\mathbb{R}}\ \rho$ where E and ρ are respectively the representation of the goal and the corresponding environment. If we look closely at $[|\_|]_{\mathbb{R}}$'s code, we can remark that:

- `scal`(k, s) is always translated as $k * s'$ where s' is the translation of s. We must therefore use the commutativity of the addition to put all the constant expression on the left side of their non-constant counterpart
- `y`(p, k) is always translated as `nth_derive`$(\texttt{sum}(\rho_p), k)$ and we must therefore:

  replace every occurrence of `sum`$(\texttt{fst}(\rho_p), pr')$ where $pr' \neq \texttt{snd}(\rho_p)$ by `sum`(ρ_p)

  replace every occurrence of `sum`$(\rho_p)(x)$ with `nth_derive`$(\texttt{sum}(\rho_p), 0)$

  replace every occurrence of `derive`$(\texttt{sum}(\rho_p))(x)$ with `nth_derive`$(\texttt{sum}(\rho_p), 1)$

This normalization is possible thanks to lemmas stating that all these substitutions are sound. Here are two examples of such lemmas:

```
Lemma nth_derive_sum_PI_0 : forall an (r1 r2 : infinite_cv_radius an),
  nth_derive (sum an r1) (D_infty_Rpser an r1 0) == sum an r2.

Lemma nth_derive_sum_PI_1 : forall an (r1 r2 : infinite_cv_radius an)
  (pr : derivable (sum an r2)),
  nth_derive (sum an r1) (D_infty_Rpser an r1 1) == derive (sum an r2) pr.
```

2.4.3 End-user tactics

The end-user of our libraries is obviously not supposed to call these tactics that deal with environment, sub-expressions, variable names, etc. We provide two friendlier tactics (namely `solve_diff_equa_on` and `solve_diff_equa`) that hide away all these details:

- `solve_diff_equa_on`(x) inspects the goal. If the goal is an equality statement,

  It quotes both sides by using `quote_side_equa` with an appropriate environment (the empty one first and the one returned by the first call then) and x; as a result, it gets two `side_equa` E_1 and E_2 and an environment *env*

  It then normalizes the equation by calling `normalize` on both sides with *env*

  The last step is to assume $[|\mathrm{E}_1 :=: \mathrm{E}_2|]_{\mathbb{N}}\ \rho$ (the user will be asked to prove this fact later) *via* `cut` and use it together with our theorem to prov the current goal
- `solve_diff_equa_on` is simply `solve_diff_equa_on`(x) where x is a fresh variable introduced by the tactic.

[6] We do not support non linear differential equations yet.

3 Applications

Thanks to this theorem and these tactics, one can prove in less than 20 lines[7] that the exponential is a solution of the equation $y^{(n+1)} = y^{(n)}$. Instead of having to deal with limits and derivatives, one just has to prove that:

$$\forall k \in \mathbb{N}, \frac{((n+1)+k)!}{k!} * \frac{1}{((n+1)+k)!} = \frac{(n+k)!}{k!} * \frac{1}{(n+k)!}$$

Without much more work, one can also prove that cosine and sine are both solutions of $y^{(2)} = -y$.

References

[1] INRIA. *The Coq proof assistant.*

[7] See `Dequa_examples` for more details on these proofs.

Appendix

We assume in the following developments that the lookup function $\texttt{nth\_error} : \forall \alpha.\, \texttt{list}\,\alpha \to \mathbb{N} \to \texttt{option}\,\alpha$ defined in the standard library is known by the reader.

Equations on power series

$$
\begin{array}{lllll}
\texttt{interp}_{\mathbb{R}} & & & : & \texttt{side\_equa} \to \texttt{list}\,\texttt{Rseq\_infty} \\
 & & & & \to \texttt{option}\,(\texttt{R} \to \texttt{R}) \\
\texttt{interp}_{\mathbb{R}} & (\texttt{cst}\ C & ,\rho) & = & \texttt{return}\ (\lambda\_.C) \\
\texttt{interp}_{\mathbb{R}} & (\texttt{scal}\ C\ E & ,\rho) & = & \texttt{interp}_{\mathbb{R}}(E,\rho)\ >\!\!>\!\!=\ (\lambda f.\,\texttt{return}(C * f)) \\
\texttt{interp}_{\mathbb{R}} & (\texttt{y}\ i\ k & ,\rho) & = & \texttt{nth\_error}(\rho, i)\ >\!\!>\!\!=\ (\lambda u_n.\,\texttt{return}(\texttt{sum}\,(u_n)^{(k)})) \\
\texttt{interp}_{\mathbb{R}} & (\texttt{opp}\ E & ,\rho) & = & \texttt{interp}_{\mathbb{R}}(E,\rho)\ >\!\!>\!\!=\ (\lambda f.\,\texttt{return}(-f)) \\
\texttt{interp}_{\mathbb{R}} & (\texttt{plus}\ E_1\ E_2 & ,\rho) & = & \texttt{interp}_{\mathbb{R}}(E_1,\rho)\ >\!\!>\!\!=\ (\lambda f_1. \\
 & & & & \texttt{interp}_{\mathbb{R}}(E_2,\rho)\ >\!\!>\!\!=\ (\lambda f_2.\,\texttt{return}(f_1 + f_2)))
\end{array}
$$

$$[|\_|]_{\mathbb{R}} : \texttt{diff\_equa} \to \texttt{list}\,\texttt{Rseq\_infty} \to \texttt{Prop}$$

$$
[|\mathrm{E}_1 :=: \mathrm{E}_2|]_{\mathbb{R}}\,\rho = \bot
\quad \text{if} \begin{cases} & \texttt{interp}_{\mathbb{R}}(E_1,\rho) = \texttt{None} \\ \text{or} & \texttt{interp}_{\mathbb{R}}(E_2,\rho) = \texttt{None} \end{cases}
$$

$$
[|\mathrm{E}_1 :=: \mathrm{E}_2|]_{\mathbb{R}}\,\rho = \forall x \in \mathbb{R}.e_1(x) = e_2(x)
\quad \text{if} \begin{cases} & \texttt{interp}_{\mathbb{R}}(E_1,\rho) = \texttt{Some}\,e_1 \\ \text{and} & \texttt{interp}_{\mathbb{R}}(E_2,\rho) = \texttt{Some}\,e_2 \end{cases}
$$

Equations on sequences over $\mathbb{R}$

$$
\begin{array}{lllll}
\texttt{interp}_{\mathbb{N}} & & & : & \texttt{side\_equa} \to \texttt{list}\ \texttt{Rseq} \to \texttt{option}\ \texttt{Rseq} \\
\texttt{interp}_{\mathbb{N}} & (\texttt{cst}\ C & ,\rho) & = & \texttt{return}\ (\texttt{An\_cst}\,(C)) \\
\texttt{interp}_{\mathbb{N}} & (\texttt{scal}\ C\ E & ,\rho) & = & \texttt{interp}_{\mathbb{N}}(E,\rho)\ >\!\!>\!\!=\ (\lambda u_n.\,\texttt{return}(C * u_n)) \\
\texttt{interp}_{\mathbb{N}} & (\texttt{y}\ i\ k & ,\rho) & = & \texttt{nth\_error}(\rho, i)\ >\!\!>\!\!=\ (\lambda u_n. \\
 & & & & \texttt{return}(\texttt{An\_nth\_deriv}\,(un, k)) \\
\texttt{interp}_{\mathbb{N}} & (\texttt{opp}\ E & ,\rho) & = & \texttt{interp}_{\mathbb{N}}(E,\rho)\ >\!\!>\!\!=\ (\lambda u_n.\,\texttt{return}(-u_n)) \\
\texttt{interp}_{\mathbb{N}} & (\texttt{plus}\ E_1\ E_2 & ,\rho) & = & \texttt{interp}_{\mathbb{N}}(E_1,\rho)\ >\!\!>\!\!=\ (\lambda u_n. \\
 & & & & \texttt{interp}_{\mathbb{N}}(E_2,\rho)\ >\!\!>\!\!=\ (\lambda v_n.\,\texttt{return}(u_n + v_n)))
\end{array}
$$

$$[|\_|]_{\mathbb{N}} : \texttt{diff\_equa} \to \texttt{list}\,\texttt{Rseq\_infty} \to \texttt{Prop}$$

$$
[|\mathrm{E}_1 :=: \mathrm{E}_2|]_{\mathbb{N}}\,\rho = \bot
\quad \text{if} \begin{cases} & \texttt{interp}_{\mathbb{N}}(E_1,\rho) = \texttt{None} \\ \text{or} & \texttt{interp}_{\mathbb{N}}(E_2,\rho) = \texttt{None} \end{cases}
$$

$$
[|\mathrm{E}_1 :=: \mathrm{E}_2|]_{\mathbb{N}}\,\rho = \forall n \in \mathbb{N}.u_n(n) = v_n(n)
\quad \text{if} \begin{cases} & \texttt{interp}_{\mathbb{N}}(E_1,\rho) = \texttt{Some}\,u_n \\ \text{and} & \texttt{interp}_{\mathbb{N}}(E_2,\rho) = \texttt{Some}\,v_n \end{cases}
$$