Symbolic integration in Prolog
Documentation

Michal Koutný

This is a documentation to the program whose task is finding a primitive functions for given function. The goal was to have a program that knows only a few pairs of function and its primitive function but can infere many pairs with just simple rules for integration.

However, there are many functions for which program is unable to find their primitive function. At least, it can serve as a demostration of integral solver for simple textbook integrals.

1 Integrating rules

Program tries to combine these rules in a compatible way to obtain the primitive function for given input.

Linearity

$$\begin{aligned}\int f(x) \pm g(x)\mathrm{d}x &= \int f(x)\mathrm{d}x \pm \int g(x)\mathrm{d}x\,,\\ \int cf(x)\mathrm{d}x &= c\int f(x)\mathrm{d}x\,,\\ \int -f(x)\mathrm{d}x &= -\int f(x)\mathrm{d}x\,,\end{aligned}$$

where c is constant with respect to x.

Per partes

If integrand can be written as a product $u'v$, we try to use integration by parts

$$\int u'v\mathrm{d}x = uv - \int uv'\mathrm{d}x\,,$$

where f' means derivative of f.

Program tries all possibilities how to factorize integrand (using only the first level product as mentioned in section about normalization) with an exception when $v = 1$.

If we obtain the same integral as we are calculating, the result is $uv/2$.

Substitution of the first type

We make some implicit assumptions about x and then we substitute x with $g(t)$, where t is new integrating variable. Possibly, we can get a form of integral that is suitable for integration with our simple methods.

$$\int f(x)\mathrm{d}x = \int f(g(t))g'(t)\mathrm{d}t = \ldots = F(t)$$

To get the result in terms of variable x, we have to know $g^{-1}(x)$, consequently, the result is $F(g^{-1}(x))$.

Substitution of the second type

In this case, we just try whether we can find a subexpression and its derivative in the formula of the function. If so, we substitute the subexpression with a new variable.

$$\int f(s(x))s'(x)\mathrm{d}x = \int f(t)\mathrm{d}t = \ldots = F(t) = F(s(x))$$

Table integrals

Table integrals are only ”absolute” knowledge of the program about primitive functions. I used following relations:

$$\begin{aligned}
\int c\mathrm{d}x &= cx\,\text{, where } c \text{ is constant,}\\
\int x\mathrm{d}x &= \frac{x^2}{2}\,,\\
\int \sin(x)\mathrm{d}x &= -\cos(x)\,,\\
\int \cos(x)\mathrm{d}x &= \sin(x)\,,\\
\int \mathrm{d}x/x &= \ln(x)\,\text{, where ln is natural logarithm,}\\
\int c^x\mathrm{d}x &= \frac{c^x}{\ln(c)}\,\text{, where } c \text{ is constant,}\\
\int x^c\mathrm{d}x &= \frac{x^{c+1}}{c+1}\,\text{, where } c \text{ is constant.}
\end{aligned}$$

2 Parts of program

Whole program consists of three quite independent parts.

2.1 Normalization

This component was originally intended as simplifier of formulas. But I needed something deterministic because I wanted two equivalent expressions to be normalized into the same form.

For normalization, I consider every formula as a sum of products, where each factor in product could be again formula of similar pattern.

Normalization predicate collapses all numeric addends/factors and also tries to reduce fractions. Multiple powers are also collapsed into one power that is product of partial powers.

Numeric addends/factors are always listed first. Same factors in one product are collapsed into one factor with appropriate power.

Little example should demostrate what normalization does.

$$1 + x + 4 - \frac{x \cdot x \sin^2(x)}{y \sin(x)} = 5 + x - \frac{x^2 \sin(x)}{y}$$

2.2 Derivative

Because finding a derivative is deterministic process, I won't explain it much in detail here. Component for derivation uses basic rules for derivatives [linearity, product, ratio, derivatives of basic functions (they implicitly use chain rule)].

Program should find derivative of any function combining elementary functiona for those table derivatives are defined in the source. (Derivatives of the type $f(x)^{g(x)}$ are found via substitution $\mathrm{e}^{g(x)\ln(f(x))}$.)

2.3 Integration

Briefly – program makes profit of Prolog backtracking ability and tries to find correct sequence of integrating methods mentioned above to find primitive function.

As backtracking whole tree would be waste of time, programs tries to use some hints to reduce amount of combinations it has to test.

2.3.1 Heuristics and pruning

Because sometimes after normalization or substitution we obtain an expression that program is not able to integrate, it uses so called ”identity substitution”, i. e. substituting a subexpression with another that is equivalent.

Trying all these identity substitutions would be very time consuming, I divided them in three groups

- those that factorize the subexpression,
- those that expands the subexpressin into the sum,

- and those that collapses subexpression sum into one term.

We use this division in the case of per partes factorization, when we only want to factorize a subexpression, not expand it into the sum etc. However, when we obtain new inegrand in per partes method, we only need expand it into the sum.

Another approach that reduces number of tested integrals is that whenever factorization via identity substitution is possible, we perform it.

A big problem of the per partes method is that it can give us difficult integrals, that we are hardly able to solve (I mean program is able), therefore, counter is used that measures ”depth” of per partes to prevent us of getting more and more complicated formulas.

Still, we can get lengthy expression hence we allow factorization with $u' = 1$ only once in a per partes chain.[1]

It is possible to get strange results using substitution of the first type (e. g. $\int x \mathrm{d}x = \sin^2(\arcsin(x))/2$), which is caused by inprudent use of the first type substitution. Therefore, program tries to test whether the use of the substitution would make any improvement in the original formula.

3 Conclusion

There are several problems that I wasn’t able to solve satisfyingly.

Firstly, program has to test everything, to be sure that it can’t really calculate the primitive function.

Connected to this is testing of the equivalence of two expressions – simple Prolog unification is too weak for this purpose, as we can get various forms of the same expression during integration process.

Whole bunch of integrals is forbidden because of missing support for rational functions. This would require programming tools for polynomial manipulation, which isn’t area in which specific features of Prolog could be effectively used.

On the other hand, the program takes advantage of Prolog backtracking as an easy way how to test possible combinations of integration rules.

I’ve also found Prolog manipulation with terms quite useful, as I needn’t to bother myself with parsing expressions or difficult term substitution.

[1]Notation for u is equal to that in section Per partes.