Design documents for new features

by the Yacas team [1]

Yacas version: 1.0.57
generated on November 28, 2006

This book contains some design documents that are in progress, for new features or features or implementation details that need to be changed and improved.

Contents

1 Introduction **2**

2 Algorithms requiring global pattern matchers **3**
- 2.1 Introduction . . . 3
- 2.2 Types of transformations . . . 3
- 2.3 Flexible pattern matching . . . 3
- 2.4 Towards a solution . . . 4
- 2.5 Composite factors . . . 5
- 2.6 Simplification in the face of non-commuting algebras . . . 6

3 How Yacas Deals With Sets of Solutions **7**
- 3.1 Introduction . . . 7
- 3.2 Implementation Semantics of Solve in Yacas . . . 7
- 3.3 Use Case Scenarios . . . 8

4 Reflection **9**

5 Multi-valued expressions **10**

6 Assume facilities **11**

7 Defining a new function in the kernel **12**

8 A user interface for Yacas **13**
- 8.1 Use case scenarios . . . 13
- 8.2 Yacas for fun . . . 13
- 8.3 Yacas for profit . . . 13
- 8.4 The current solution . . . 13
- 8.5 Possible additional ways to offer information . . . 13
- 8.6 Other facilities . . . 14

9 The static code analyzer **15**

10 GNU Free Documentation License **16**

Chapter 1

Introduction

This book contains some design documents that are in progress, for new features or features or implementation details that need to be changed and improved.

Chapter 2

Algorithms requiring global pattern matchers

2.1 Introduction

The default YACAS transformation rules work on expressions locally. For instance, one could write a transformation rule to replace $1x$ with x. This facility is not suitable for transformation rules that require a more global scope. For instance, this facility does not easily allow one to transform an expression $abca$ to a^2bc.

This type of transformation is often required, and this chapter discusses this issue. The types of transformations required will be discussed, along with possible implementation schemes.

2.2 Types of transformations

Selections within terms

A term is an expression with only factors, subexpressions multiplied. Its general form is $a_1...a_n$ for n factors.

The first, basic type of expression one would like to be able to match is an expression like $...a^n...a^m...$, which in its most general form discovers that an expression only contains multiplications, and that it contains a factor a^n and a factor a^m. We would like this expression to be changed into $...a^{n+m}....$

Another example is simplification using trigonometric identities, where a term containing two trigonometric factors is reduced to two terms with each one trigonometric factor. Identities like

$$\cos u \sin v = \frac{1}{2}\left(\sin\left(v-u\right)+\sin\left(v+u\right)\right)$$

are easy to write down in one line. Ideally, YACAS would allow a programmer to write this down in one line and then have a global pattern matcher recognize when this pattern is applicable, and apply the transformation rule.

Other examples include:

$$\frac{a^n}{a^m} = a^{n-m}$$

$$\exp\left(x\right)\exp\left(y\right) = \exp\left(x+y\right)$$

and for a given integer k:

$$\frac{(n+k)!}{n!} = \left(n+k\right)\left(n+k-1\right)...\left(n+1\right)$$

These are all patterns that could be found within one term containing only factors.

Selections across terms

In addition to being able to match a pattern within one term, it is sometimes useful to be able to recognize patterns across multiple terms. One example is

$$\exp\left(x\right)+2\exp\left(\frac{x}{2}\right)+1$$

In this case, it is possible to write this expression as:

$$\left(\exp\left(\frac{x}{2}\right)\right)^2+2\exp\left(\frac{x}{2}\right)+1=0$$

which in turn can be simplified further, by setting

$$y=\exp\left(\frac{x}{2}\right)$$

yielding

$$\left(y+1\right)^2=0$$

and thus $y=-1$, $\exp\left(\frac{x}{2}\right)=-1$ after factoring. It then follows that the solution is $x=\imath\cdot 2\pi$:

```
In> Solve((Exp(x/2)+1)^2==0,x)
Out> Complex(0,2*Pi);
In> f(x):=N(Exp(x)+2*Exp(x/2)+1)
Out> True;
In> f(Complex(0,2*Pi))
Out> 0;
```

2.3 Flexible pattern matching

In addition to using more flexible, global pattern matching facilities to simplify expressions, it is sometimes necessary to use such more global pattern matchers for other algorithms also.

For instance, for trigonometric identities, there is sometimes an exact result if the argument is a rational number times π. One would like to be able to recognize $\frac{\pi}{2}$, $\frac{1}{2}\pi$, $\pi\frac{1}{2}$, $4\pi\frac{1}{8}$, and perhaps even 0.5π as being half Pi.

With integration, there is often an antiderivative for an expression if an argument can be written as $ax + b$ where x is the variable being integrated over, and a and b are constants that do not depend on x.

2.4 Towards a solution

As soon as this facility is available, it is expected to be used frequently, and deep in the system. The consequence is that it should be implemented efficiently.

Furthermore, in keeping with the rest of the design of the language, it should be possible to write only a few lines of code in order to specify patterns to be matched and transformations to be performed.

Steps local transformation rules are suited for

Before matching global patterns, the local transformation rules (which are efficient) can be used to do preliminary cleaning up of expressions. Expressions like $a\,(b+c)\,d$ can easily be converted to $abd + acd$, or $\frac{a}{\frac{b}{c}}$ can be easily converted to $\frac{ac}{b}$.

Patterns within terms

For patterns within terms, it is necessary to gather the relevant information from the term in order to be able to decide if a pattern matches. Suppose we want to recognize an expression of the form described in the beginning of this chapter:

$...a^n...a^m...$.

One solution is to do it in scripts. One needs to gather all the factors. The first option is to write a custom bit of code for this type of transformation, where all factors are gathered in a list, combining the symbol with its exponent. While traversing the expression, one adds items to this list.

For instance for an expression ab^2cda one would first add `(a,1)`, then `(b,2)`, `(c,1)` and `(d,1)`. Lastly, for the last factor, a, the system could notice that it was already in the list, and just update this item to `(a,2)`. One would end up with the set `((a,2),(b,2),(c,1),(d,1))`. This could be called an internal representation.

After this process finishes, one can traverse this list, producing the required result a^2b^2cd.

This process can be generalized. The steps involved are:

1. traverse the expression to gather information (building an internal representation).
2. for each item, combine it with the internal database of information already gathered. The internal database for this expression will change. The way it will change depends on the type of simplification needed.
3. The internal database representing the required information from the expression is finished. It is an internal representation, however, so it needs to be converted back to a normal expression.

The above scheme is flexible, in that the specific method of adding a sub-expression to the already existing database can be customized for different types of simplification or pattern matching. transformations could even be done in-place. The specific algorithm for adding a term to the database then acts as a filter.

The difficult part is to choose the form of the database, as that ultimately defines the types of transformations and simplifications that are possible.

When only transformations within one term are needed, the situation is not too difficult.

$$(a[1][1]*...*a[1][n1])/(b[1][1]*...*b[1][m1])+...+(a[p][1]*...*a[p][np])/(b[p][1]*$$

Eg. one could write an expression as a set of terms, where each term is a rational function, consisting of a numerator and denominator, each with just simple factors. One could then process each term, and maintain a database for each term. Thus one could readily simplify $\frac{a^n}{a^m}$ to a^{n-m}, or $\frac{(n+1)!}{n!}$ to $n+1$. The trigonometric simplification scheme described earlier in this chapter would be more sophisticated, as it would require changing one term to two (as a multiplication within a term is changed into an addition of two terms).

Specifying transformation rules in the simplification scheme

The algorithm described in the simplification section requires that factors be added to the database of information on the term the factor belongs to one at a time.

For instance, the database might already have an entry representing a^n. When a new term a^m is encountered, one would want to combine these two into a^{n+m}. This is easy to do with custom code that can directly access the database internal representation.

At the same time, one would ideally want to specify this transformation using notation similar to

$$a^n a^m = a^{n+m}$$

In stead of writing custom hand-written code for this transformation, one would want a system were the a^n part is matched in the database and the a^m part in the expression being processed. The result, a^{n+m}, would be the result, and one would want the system to remove the a^n from the database and put back a^{n+m}.

Actually, for efficiency reasons, one would want the system to change a^n to a^{n+m} in the database. But this is a very specific example where this is easily possible. For transformation rules like

$$\frac{(n+k_{\mathrm{InInteger}})!}{n!} = (n+k)\ldots(n+1)$$

one needs a database for a rational function, and one needs to remove one factor and replace it with many other factors.

In the case of the trigonometric identities, one needs to replace something of the form AB to something of the form $C+D$, which requires actually removing the actual term at hand (and not just one factor) and replace it with two terms.

One option is to have the $a^n a^m = a^{n+m}$ simplification be performed automatically, then convert the entire expression into internal representation with one additional facility in the internal representation that one additional collection is made; one can collect factors or terms that one knows will be relevant for the simplification. when simplifying factorials, one can collect all factorials in a special part of the internal representation so they are all grouped together. The algorithm can then proceed to process only these parts. For instance,

$$\frac{(n+2)!ab}{cn!}$$

could be converted to

$$(n+2)!\,(n!)^{-1}\,abc^{-1}$$

So that the left side part can be further simplified, resulting in

$$(n+2)\,(n+1)\,abc^{-1}$$

Dealing with rational functions

A problem arises with rational functions. In the algorithm described above for traversing an expression, gathering information, the fact that the numerator and denominator of the term being examined can have a greatest common denominator which does not equal one, has not been taken into account.

This can be a problem when the expression is brought into the form described above, as this normalization step would first convert $\frac{a+b}{c}$ to $\frac{a}{c}+\frac{b}{c}$. This breaks up the expression. If $a+b$ and c had common factors, this is now lost.

Thus the rational expression first needs to be scanned for greatest common divisors before being split up.

The problem of finding greatest common divisors is a little bit more complicated than described here. For instance the following expression

$$\frac{x}{x^2-1}+a+\frac{1}{x^2-1}$$

is actually equivalent to

$$a+\frac{1}{x-1}$$

This can only be discovered if the first and the last term are combined into one, leaving out the middle term. Perhaps the fact that the two denominators have a greatest common divisor not equal to one can be used to find terms that can be combined.

Another option is to not split up rational terms when there is an addition involved in the denominator, as little is expected to be gained from the simplification any way after removing greatest common divisors. In stead, the numerator and denominator could be simplified as sub-expressions in their own simplification step.

Global pattern matching in general

Finding patterns in terms has been described above. The idea presented was to have a generic database format in which to collect information about the expression, and then have a routine traverse the expression, applying a filter to each term, bringing all terms to the database.

This scheme might not be the most suitable solution when matching patterns for use in other algorithms. In such cases one is generally interested in whether an expression can possibly be brought into a specific form, which does not necessarily match the form the simplifier brings the expression in.

Suppose we want to find out if an expression matches the form $ax+b$ where both a and b don't depend on x. Further more, the algorithm is likely to need the actual values for a and b.

If the algorithm for simplification described earlier were used, one would end up with a list of terms, each with lists of factors in the intermediate format. The information that is needed would be missing. For example:

$$A+B+Cx+D+Ex+F$$

would have to be written down as

$$(C+E)\,x+A+B+D+F$$

before one can recognize that $ax+b$ fits, and find the exact forms for a and b.

In a sense the method for arriving at this stage is almost the same: first gather information, and then return the information in the requested form. The problem is that this feature, global pattern matching, requires more flexibility.

The fact that the algorithms and internal representation of the simplification scheme described above are not suited for this task can be seen from the following example; suppose we wanted to match $ax+b$ to the expression

$$\frac{A+x+B}{E+D}+E$$

One would want this expression written out as

$$\frac{1}{E+D}x+\frac{A+B}{E+D}+E$$

so that the constants can readily be identified. However, as discussed in the section on rational terms, for simplification it is not wise to split up such rational terms. It is required for this example, however. The filtering step could be made more sophisticated so that this operation can be dealt with.

A more serious concern is with efficiency. Apart from splitting up the expression in too small chunks (one only needs a and b, and thus would only need to gather these), there is another danger: the conversion to internal representation continues until the full expression is processed.

In practice, while the expression is being processed, if custom code were written, the matcher could decide *while* matching, that the pattern will never fit, and terminate immediately. For example, a and b should be independent of x, so if a factor like $\sin x$ is encountered the matching can stop immediately.

If the scheme described in the section dealing with simplification is to be used for general pattern matching also, another step needs to be added to it; the filter operation should be able to terminate the process when it decides early on that code further upstream will fail any way.

Other tools at our disposal

The procedure for simplification offers some interesting opportunities. Apart from the filter returning early on whether it succeeded or failed, one could already have the filter perform transformations on (parts of) the expression, and return the transformed and untransformed parts.

2.5 Composite factors

One important issue with simplification is normalization. Factors can be functions in stead of simple variables. In order to decide that $\sin ab$ equals $\sin ba$ and thus simplify $\sin ab-\sin ba$ to zero, it suffices for the system to bring expressions to normal form.

In this case, if $\sin ba$ were first converted to $\sin ab$, the system could readily collect terms and discover that the terms described above cancel each other out and result in zero.

For this, the system needs to impose an order for factors. For this, `LessThan` can be used. `LessThan` defines an ordering for atoms:

```
In> LessThan(a,b)
Out> True;
In> LessThan(b,a)
```

```
Out> False;
In> LessThan(2,a)
Out> True;
In> LessThan(2,1)
Out> False;
In> LessThan(a,"b")
Out> False;
```

Ordering for composed expressions can be created based on the `LessThan` function.

2.6 Simplification in the face of non-commuting algebras

In non-commuting algebras, the rule

$$AB = BA$$

does not hold. This has consequences for a system that is trying to simplify the expression passed in. When gathering information to be stored in the database, what the operation is in fact doing is moving factors around until they are next to each other and can be combined. For instance,

$$A^2BCA$$

would first be re-ordered to

$$A^2ABC$$

which then readily converts to

$$A^3BC$$

Extending this system to also support non-commuting algebra involves disallowing these swaps. For groups of symbols, one could specify that two symbols do not commute. For the above example, supposing A and B do not commute, the resulting expression could then terminate as:

$$A^2BAC$$

One step further would be to support commutation relations. Suppose that after one established that A and B do not commute, that the following relation holds:

$$AB - BA = C$$

Then the part of the expression BA could be converted to $AB - C$. This could then result in the expression:

$$A^2 (AB - C) C$$

Which then can be written as:

$$A^3B - A^2C^2$$

Factoring could then yield:

$$A^2 \left(AB - C^2\right)$$

A potential risk is with the global transformation rules, which do not necessarily adhere to the rules for non-commuting algebras. A special non-commuting multiplication operator could be defined for this to guarantee that this is never a problem.

Chapter 3

How Yacas Deals With Sets of Solutions

3.1 Introduction

(This is a draft)

Worries:

- need to change all code that uses Solve
- need a lot of changes in documentation
- arguments to solve are a bit more verbose than the previous version: Solve(`eq1,eq2`,vars) versus Solve(eq1 And eq2,vars). Suddenly things like Solve(leftlist==rightlist,vars) is not possible any more. This has to be done with extra commands (which is ok?). It can not stay the way it was, because lists now mean something else, a collection of disjunct solutions.

The difference between a problem stated and a solution given is a subtle one. From a mathematical standpoint,

```
In> Integrate(x,0,B)Cos(x)
Out> Sin(B);
```

And thus

```
Integrate(x,0,B)Cos(x) == Sin(B)
```

is a true statement. Furthermore, the left hand side is mathematically equivalent to the right hand side. Working out the integration, to arrive at an expression that doesn't imply integration any more is generally perceived to be a more desirable result, even though the two sides are equivalent mathematically.

This implies that the statement of a set of equations declaring equalities is on a same footing as the resulting equations stating a solution:

$$ax+b=c \Rightarrow x=\frac{c-b}{a}.$$

If the value of x is needed, the right hand side is more desirable.

Viewed in this way, the responsibility of a `Solve` function could be to manipulate a set of equations in such a way that a certain piece of information can be pried from it (in this case the value of $x=x\left(a,b,c\right)$.

A next step is to be able to use the result returned by a `Solve` operation.

3.2 Implementation Semantics of Solve in Yacas

Suppose there is a set of variables that has a specific combination of solutions and these solutions need to be filled in in an expression: the `Where` operator can be used for this:

```
In> x^2+y^2 Where x==2 And y==3
Out> 13;
```

Solve can return one such solution tuple, or a list of tuples. The list of equations can be passed in to Solve in exactly the same way. Thus:

```
In> Solve(eq1,var)
Out> a1==b1;
In> Solve(eq1 And eq2 And eq3,varlist)
Out> {a1==b1 And a2==b2,a1==b3 And a2==b4};
```

These equations can be seen as simple simplification rules, the left hand side showing the old value, and the right hand side showing the new value. Interpreted in that way, `Where` is a little simplifier for expressions, using values found by Solve.

Assigning values to the variables values globally can be handled with an expression like

```
solns := Solve(equations,{var1,var2});
{var1,var2} := Transpose({var1,var2} Where solns);
```

Multiple sets of values can be applied:

```
In> x^2+y^2 Where {x==2 And y==2,x==3 And y==3}
Out> {8,18};
```

This assigns the the variables lists of values. These variables can then be inserted into other expressions, where threading will fill in all the solutions, and return all possible answers.

Groups of equations can be combined, with

```
Equations := EquationSet1 AddTo EquationSet2
```

or,

```
Equations := Equations AddTo Solutions;
```

Where `Solutions` could have been returned by `Solve`. This last step makes explicit the fact that equations are on a same footing, mathematically, as solutions to equations, and are just another way of looking at a problem.

The equations returned can go farther in that multiple solutions can be returned: if the value of x is needed and the equation determining the value of x is $x\equiv\left|a\right|$, then a set of returned solutions could look like:

```
Solutions := { a>=0 And x==a, a<0 And x== -a }
```

The semantics of this list is:

```
either a >= 0 And x equals a, or
       a < 0 And x equals -a
```

When more information is published, for instance the value of a has been determined, the sequence for solving this can look like:

```
In> Solve(a==2 AddTo Solutions,{x})
Out> x==2;
```

The solution `a<0 And x==-a` can not be satisfied, and thus is removed from the list of solutions.

Introducing new information can then be done with the AddTo operator:

```
In> Solutions2 := (a==2 AddTo Solutions);
Out> { a==2 And a>=0 And x==a, a==2
  And a<0 And x==-a };
```

In the above case both solutions can not be true any more, and thus when passing this list to `Solve`:

```
In> Solve(Solutions2,{x})
Out> x==2;
```

`AddTo` combines multiple equations through a tensor-product like scheme:

```
In> {A==2,c==d} AddTo {b==3 And  d==2}
Out> {A==2 And b==3 And d==2,c==d
  And b==3 And d==2};
In> {A==2,c==d} AddTo {b==3, d==2}
Out> {A==2 And b==3,A==2 And d==2,c==d
  And b==3,c==d And d==2};
```

A list `a,b` means that a is a solution, OR b is a solution. AddTo then acts as a AND operation:

```
(a or b) and (c or d) =>
(a or b) Addto (c or d) =>
(a and c) or (a and d) or (b and c) or (b and d)
```

`Solve` gathers information as a list of identities. The second argument is a hint as to what it needs to solve for. It can be a list of variables, but also "Ode" (to solve ordinary differential equations), "Trig" (to simplify for trigonometric identities), "Exp" to simplify for expressions of the form $\exp(x)$, or "Logic" to simplify expressions containing logic. The "Logic" simplifier also should deal with $a > 2 \wedge a < 0$ which it should be able to reduce to `False`.

`Solve` also leaves room for an 'assume' type mechanism, where the equations evolve to keep track of constraints. When for instance the equation $x = \sin y$ is encountered, this might result in a solution set

```
y == ArcSin(x) And x>=-1 And x <= 1
```

3.3 Use Case Scenarios

To be filled in

Chapter 4

Reflection

Chapter 5

Multi-valued expressions

Chapter 6

Assume facilities

Chapter 7

Defining a new function in the kernel

This section will explain how to add a new kernel-level function to Yacas, and will explain why it is this way.

Chapter 8

A user interface for Yacas

In practice, for power users, Yacas is already a convenient system when accessible from the command line. The command line allows one to enter calculations rapidly when the user already knows the commands that are available, and knows what he wants to do and what is possible. This unfortunately includes all developers working on Yacas, so there is little incentive to create a user interface front end for Yacas.

Nevertheless, this chapter will try to specify a graphical user interface that services users other than power users.

The single big usability issue is currently that one already needs to know the system a bit before one can use it. The user is greeted with an intimidating flashing cursor, waiting for the user to enter a command. The user needs to know which commands are available, and when to use those commands. In short, the user, whether it is a new or experienced user, can be greatly helped with information that is more readily available.

8.1 Use case scenarios

There are two use case scenarios the author can think of:

1. The user wants to have some fun, and enjoys playing around with math.
2. The user has a specific calculation that needs to be performed, and hopes Yacas can do it quickly.

The following sections describe the possible features a graphical front end could offer to facilitate these.

8.2 Yacas for fun

When a user is bored and wants to be entertained, and when entertainment includes having fun with math, the user might start up the (currently hypothetical) graphical user interface and start to explore the possibilities. The first question the user will ask is "What can Yacas do?", and when an interesting subject is found, perhaps play with it, entering various parameters to a calculation model, generate graphs, and maybe even learn something new along the way.

8.3 Yacas for profit

When the user has a real world problem, a calculation that needs to be performed, one that he knows he can do perhaps by looking it up in a book or writing dozens of pages until he reaches the result, or perhaps verify that a calculation is correct, the goal is much narrower. The user already knows what he wants to do, but might want to know *how* he can do it.

8.4 The current solution

With the command line version of Yacas, the solution for the above mentioned use case scenarios is to read the manual, and perhaps try some examples until the user understands the tools available.

The user can then play around with some commands, and finally set up a file with code that will perform the calculation and run that file.

A graphical user interface offers the opportunity to allow the user to access relevant information more quickly than scanning the hundreds of pages of documentation (with the chance of getting lost).

Of course documentation in combination with examples that show how to do specific types of calculations go a long way when a user needs to find out how to do a specific calculation, or wants to know what is possible. However, a user interface might provide the required information more readily.

8.5 Possible additional ways to offer information

"What is available?" information

When typing in a command, it happens often (even with experienced users) that the user forgot the exact arguments to that function, or the order of the arguments.

One solution might be to pop up a tip box with the possible ways the command can be completed.

When the user doesn't know which command to use in the first place, the system could provide a list of possible commands, perhaps categorized by type of operation. For each command a short blurb could be shown about what the routine does, when it is applicable, and perhaps some examples for the user to try out to get a handle on the command.

Arguably the last option is offered by the manual already.

"How do I ...?" information

For more elaborate use, the user might be better off with example calculations. A well-documented example showing how a calculation is done goes a long way.

The user can then use the example as a template for his own calculation. The user interface could even offer a facility to have a template for a calculation, where the user enters some final parameters for that specific calculation. With such a template, most of the work is already done, and all the user needs to do is change some parameters to reflect the calculation he wants to do.

8.6 Other facilities

In addition to the features described above, where a graphical front end offers information in a more flexible way, there are other facilities a graphical front end could offer.

Repeated calculations

Computers are good at doing repetitive tasks. If some task is performed often, it might be a good idea to extend the "template example with parameters" model to actually allow the user to design a user interface for a specific calculation so a calculation using that model can be entered more quickly. Enter a few parameters, and out come the numbers and graphs.

Facilities for programmers

For developers, a good debugger could be handy. The usual facilities like putting breakpoints, stepping through code, seeing (the values of) local variables, could be handy. The command line version already offers a useful command line interactive debugger, so this feature might not be too important.

A tree view of the source code, allowing a programmer to easily navigate through the code could be useful. As a project (and the code body) becomes larger and larger it becomes harder to find things in the scripts.

Chapter 9

The static code analyzer

Yacas has some tools to assess the quality of the scripts. The code checking tools are never finished, as new bugs are found, and guards against them added.

The idea behind the static code checkers is to check that coding standards are upheld, and to mark code that is dangerous and thus is likely to be buggy.

The following sections each describe one specific type of test. The static code analysis code can be found in `codecheck.rep` .

Interface check

As described in an essay elsewhere, files should be careful with what they expose to the environment. the `def` file mechanism and the `LocalSymbols` routine should be used for this. The `interface` check verifies that no global resources are accidentally

The rules that should be upheld are:

1. global variables should not be accessible to the outside world. They should be made local to the module by using `LocalSymbols`.
2. functions can be global, exposed to the outside world, iff they are declared in the corresponding `def` file. Otherwise, they should be made local to the module with `LocalSymbols`.
3. files should not be loaded with `Load` or `Use`, explicitly. Rather, the module should depend on the system automatically loading the right file through the `def` file mechanism.
4. scripts in the standard library should just contain simple function definitions, transformation rules, and initializations of global variables local to that module.

The `interface` check also assumes the code to consist of simple function definitions. It is meant to be used for the scripts in the standard scripts library. Exposing functionality to the outside world is usually less of a problem in one-off scripts to do specific calculations, for instance.

General rules

The following rules are general guide lines.

1. the static code analyzer should be able to recognize, when possible, if variables or functions are defined but not used.
2. functions or variables that are not declared anywhere should be reported, as it might be a typing error. Yacas evaluation just skips the function call, or lets the variable evaluate to itself.
3. platform-specific code is not allowed, in general, and thus use of SystemCall or PlatformOS or related variables or functions should be reported.
4. macro-like functions with local variables, which use `Eval`, or `Map` or related functions that re-evaluate an expression, should be flagged if the variables are not made unique through `LocalSymbols`.
5. the tools could discover if a variable is masked by another local variable with the same name.

Transformation rule checks

1. ideally there should be a way to determine that type changes are made. For instance, `0*{a,b,c}` should return a list, `{0,0,0}` , and not zero.
2. rules that are completely masked by other rules, and can never be reached, should be flagged.
3. rules with the same precedence, but which are ambiguous, should be reported, as the order in which they are tried might have impact on the result of application of the transformation rules.
4. some form of type checking might be possible, by declaring input types of various functions, and then examining the surrounding code to detect when incorrect types might be passed in. It seems to happen a lot that functions don't verify their input.

Coverage checks

Next to checking if functions are declared in the associated `def` file, the analyzer could also detect if:

1. there is no test code for the function
2. there is no or incorrect documentation for the function

Chapter 10

GNU Free Documentation License

Preamble

The purpose of this License is to make a manual, textbook, or other written document "free" in the sense of freedom: to assure everyone the effective freedom to copy and redistribute it, with or without modifying it, either commercially or noncommercially. Secondarily, this License preserves for the author and publisher a way to get credit for their work, while not being considered responsible for modifications made by others.

This License is a kind of "copyleft", which means that derivative works of the document must themselves be free in the same sense. It complements the GNU General Public License, which is a copyleft license designed for free software.

We have designed this License in order to use it for manuals for free software, because free software needs free documentation: a free program should come with manuals providing the same freedoms that the software does. But this License is not limited to software manuals; it can be used for any textual work, regardless of subject matter or whether it is published as a printed book. We recommend this License principally for works whose purpose is instruction or reference.

Applicability and Definitions

This License applies to any manual or other work that contains a notice placed by the copyright holder saying it can be distributed under the terms of this License. The "Document", below, refers to any such manual or work. Any member of the public is a licensee, and is addressed as "you".

A "Modified Version" of the Document means any work containing the Document or a portion of it, either copied verbatim, or with modifications and/or translated into another language.

A "Secondary Section" is a named appendix or a front-matter section of the Document that deals exclusively with the relationship of the publishers or authors of the Document to the Document's overall subject (or to related matters) and contains nothing that could fall directly within that overall subject. (For example, if the Document is in part a textbook of mathematics, a Secondary Section may not explain any mathematics.) The relationship could be a matter of historical connection with the subject or with related matters, or of legal, commercial, philosophical, ethical or political position regarding them.

The "Invariant Sections" are certain Secondary Sections whose titles are designated, as being those of Invariant Sections, in the notice that says that the Document is released under this License.

The "Cover Texts" are certain short passages of text that are listed, as Front-Cover Texts or Back-Cover Texts, in the notice that says that the Document is released under this License.

A "Transparent" copy of the Document means a machine-readable copy, represented in a format whose specification is available to the general public, whose contents can be viewed and edited directly and straightforwardly with generic text editors or (for images composed of pixels) generic paint programs or (for drawings) some widely available drawing editor, and that is suitable for input to text formatters or for automatic translation to a variety of formats suitable for input to text formatters. A copy made in an otherwise Transparent file format whose markup has been designed to thwart or discourage subsequent modification by readers is not Transparent. A copy that is not "Transparent" is called "Opaque".

Examples of suitable formats for Transparent copies include plain ASCII without markup, Texinfo input format, LaTeX input format, SGML or XML using a publicly available DTD, and standard-conforming simple HTML designed for human modification. Opaque formats include PostScript, PDF, proprietary formats that can be read and edited only by proprietary word processors, SGML or XML for which the DTD and/or processing tools are not generally available, and the machine-generated HTML produced by some word processors for output purposes only.

The "Title Page" means, for a printed book, the title page itself, plus such following pages as are needed to hold, legibly, the material this License requires to appear in the title page. For works in formats which do not have any title page as such, "Title Page" means the text near the most prominent appearance of the work's title, preceding the beginning of the body of the text.

Verbatim Copying

You may copy and distribute the Document in any medium, either commercially or noncommercially, provided that this License, the copyright notices, and the license notice saying this License applies to the Document are reproduced in all copies, and that you add no other conditions whatsoever to those of this License. You may not use technical measures to obstruct or control the reading or further copying of the copies you make or distribute. However, you may accept compensation in exchange for copies. If you distribute a large enough number of copies you must also follow the conditions in section 3.

You may also lend copies, under the same conditions stated above, and you may publicly display copies.

Copying in Quantity

If you publish printed copies of the Document numbering more than 100, and the Document's license notice requires Cover Texts, you must enclose the copies in covers that carry, clearly and legibly, all these Cover Texts: Front-Cover Texts on the front cover, and Back-Cover Texts on the back cover. Both covers must also clearly and legibly identify you as the publisher of these copies. The front cover must present the full title with all words of the title equally prominent and visible. You may add other material on the covers in addition. Copying with changes limited to the covers, as long as they preserve the title of the Document and satisfy these conditions, can be treated as verbatim copying in other respects.

If the required texts for either cover are too voluminous to fit legibly, you should put the first ones listed (as many as fit reasonably) on the actual cover, and continue the rest onto adjacent pages.

If you publish or distribute Opaque copies of the Document numbering more than 100, you must either include a machine-readable Transparent copy along with each Opaque copy, or state in or with each Opaque copy a publicly-accessible computer-network location containing a complete Transparent copy of the Document, free of added material, which the general network-using public has access to download anonymously at no charge using public-standard network protocols. If you use the latter option, you must take reasonably prudent steps, when you begin distribution of Opaque copies in quantity, to ensure that this Transparent copy will remain thus accessible at the stated location until at least one year after the last time you distribute an Opaque copy (directly or through your agents or retailers) of that edition to the public.

It is requested, but not required, that you contact the authors of the Document well before redistributing any large number of copies, to give them a chance to provide you with an updated version of the Document.

Modifications

You may copy and distribute a Modified Version of the Document under the conditions of sections 2 and 3 above, provided that you release the Modified Version under precisely this License, with the Modified Version filling the role of the Document, thus licensing distribution and modification of the Modified Version to whoever possesses a copy of it. In addition, you must do these things in the Modified Version:

1. Use in the Title Page (and on the covers, if any) a title distinct from that of the Document, and from those of previous versions (which should, if there were any, be listed in the History section of the Document). You may use the same title as a previous version if the original publisher of that version gives permission.
2. List on the Title Page, as authors, one or more persons or entities responsible for authorship of the modifications in the Modified Version, together with at least five of the principal authors of the Document (all of its principal authors, if it has less than five).
3. State on the Title page the name of the publisher of the Modified Version, as the publisher.
4. Preserve all the copyright notices of the Document.
5. Add an appropriate copyright notice for your modifications adjacent to the other copyright notices.
6. Include, immediately after the copyright notices, a license notice giving the public permission to use the Modified Version under the terms of this License, in the form shown in the Addendum below.
7. Preserve in that license notice the full lists of Invariant Sections and required Cover Texts given in the Document's license notice.
8. Include an unaltered copy of this License.
9. Preserve the section entitled "History", and its title, and add to it an item stating at least the title, year, new authors, and publisher of the Modified Version as given on the Title Page. If there is no section entitled "History" in the Document, create one stating the title, year, authors, and publisher of the Document as given on its Title Page, then add an item describing the Modified Version as stated in the previous sentence.
10. Preserve the network location, if any, given in the Document for public access to a Transparent copy of the Document, and likewise the network locations given in the Document for previous versions it was based on. These may be placed in the "History" section. You may omit a network location for a work that was published at least four years before the Document itself, or if the original publisher of the version it refers to gives permission.
11. In any section entitled "Acknowledgements" or "Dedications", preserve the section's title, and preserve in the section all the substance and tone of each of the contributor acknowledgements and/or dedications given therein.
12. Preserve all the Invariant Sections of the Document, unaltered in their text and in their titles. Section numbers or the equivalent are not considered part of the section titles.
13. Delete any section entitled "Endorsements". Such a section may not be included in the Modified Version.
14. Do not retitle any existing section as "Endorsements" or to conflict in title with any Invariant Section.

If the Modified Version includes new front-matter sections or appendices that qualify as Secondary Sections and contain no material copied from the Document, you may at your option designate some or all of these sections as invariant. To do this, add their titles to the list of Invariant Sections in the Modified Version's license notice. These titles must be distinct from any other section titles.

You may add a section entitled "Endorsements", provided it contains nothing but endorsements of your Modified Version by various parties – for example, statements of peer review or that the text has been approved by an organization as the authoritative definition of a standard.

You may add a passage of up to five words as a Front-Cover Text, and a passage of up to 25 words as a Back-Cover Text, to the end of the list of Cover Texts in the Modified Version. Only one passage of Front-Cover Text and one of Back-Cover Text may be added by (or through arrangements made by) any one entity. If the Document already includes a cover text for the same cover, previously added by you or by arrangement made by the same entity you are acting on behalf of, you may not add another; but you may replace the old one, on explicit permission from the previous publisher that added the old one.

The author(s) and publisher(s) of the Document do not by this License give permission to use their names for publicity for or to assert or imply endorsement of any Modified Version.

Combining Documents

You may combine the Document with other documents released under this License, under the terms defined in section 4 above for modified versions, provided that you include in the combination all of the Invariant Sections of all of the original documents, unmodified, and list them all as Invariant Sections of your combined work in its license notice.

The combined work need only contain one copy of this License, and multiple identical Invariant Sections may be replaced with a single copy. If there are multiple Invariant Sections with the same name but different contents, make the title of each such section unique by adding at the end of it, in parentheses, the name of the original author or publisher of that section if known, or else a unique number. Make the same adjustment to the section titles in the list of Invariant Sections in the license notice of the combined work.

In the combination, you must combine any sections entitled "History" in the various original documents, forming one section entitled "History"; likewise combine any sections entitled "Acknowledgements", and any sections entitled "Dedications". You must delete all sections entitled "Endorsements."

Collections of Documents

You may make a collection consisting of the Document and other documents released under this License, and replace the individual copies of this License in the various documents with a single copy that is included in the collection, provided that you follow the rules of this License for verbatim copying of each of the documents in all other respects.

You may extract a single document from such a collection, and distribute it individually under this License, provided you insert a copy of this License into the extracted document, and follow this License in all other respects regarding verbatim copying of that document.

Aggregation With Independent Works

A compilation of the Document or its derivatives with other separate and independent documents or works, in or on a volume of a storage or distribution medium, does not as a whole count as a Modified Version of the Document, provided no compilation copyright is claimed for the compilation. Such a compilation is called an "aggregate", and this License does not apply to the other self-contained works thus compiled with the Document, on account of their being thus compiled, if they are not themselves derivative works of the Document.

If the Cover Text requirement of section 3 is applicable to these copies of the Document, then if the Document is less than one quarter of the entire aggregate, the Document's Cover Texts may be placed on covers that surround only the Document within the aggregate. Otherwise they must appear on covers around the whole aggregate.

Translation

Translation is considered a kind of modification, so you may distribute translations of the Document under the terms of section 4. Replacing Invariant Sections with translations requires special permission from their copyright holders, but you may include translations of some or all Invariant Sections in addition to the original versions of these Invariant Sections. You may include a translation of this License provided that you also include the original English version of this License. In case of a disagreement between the translation and the original English version of this License, the original English version will prevail.

Termination

You may not copy, modify, sublicense, or distribute the Document except as expressly provided for under this License. Any other attempt to copy, modify, sublicense or distribute the Document is void, and will automatically terminate your rights under this License. However, parties who have received copies, or rights, from you under this License will not have their licenses terminated so long as such parties remain in full compliance.

Future Revisions of This License

The Free Software Foundation may publish new, revised versions of the GNU Free Documentation License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns. See `http://www.gnu.org/copyleft/`.

Each version of the License is given a distinguishing version number. If the Document specifies that a particular numbered version of this License "or any later version" applies to it, you have the option of following the terms and conditions either of that specified version or of any later version that has been published (not as a draft) by the Free Software Foundation. If the Document does not specify a version number of this License, you may choose any version ever published (not as a draft) by the Free Software Foundation.

ADDENDUM: How to use this License for your documents

To use this License in a document you have written, include a copy of the License in the document and put the following copyright and license notices just after the title page:

```
Copyright (C) YEAR   YOUR NAME. Permission is
granted to copy, distribute and/or modify this
document under the terms of the GNU Free
Documentation License, Version 1.1 or any later
version published by the Free Software Foundation;
with the Invariant Sections being LIST THEIR
TITLES, with the Front-Cover Texts being LIST, and
with the Back-Cover Texts being LIST. A copy of
the license is included in the section entitled
``GNU Free Documentation License''.
```

If you have no Invariant Sections, write "with no Invariant Sections" instead of saying which ones are invariant. If you have no Front-Cover Texts, write "no Front-Cover Texts" instead of "Front-Cover Texts being LIST"; likewise for Back-Cover Texts.

If your document contains nontrivial examples of program code, we recommend releasing these examples in parallel under your choice of free software license, such as the GNU General Public License, to permit their use in free software.