Guarded Expressions in Practice

Andreas Dolzmann and Thomas Sturm

Fakultät für Mathematik und Informatik
Universität Passau
D-94030 Passau, Gemany

E-mail: {dolzmann,sturm}@fmi.uni-passau.de

January 21, 1997

MIP-9702

Abstract

Computer algebra systems typically drop some degenerate cases when evaluating expressions, e.g., x/x becomes 1 dropping the case $x = 0$. We claim that it is feasible in practice to compute also the degenerate cases yielding *guarded expressions*. We work over real closed fields but our ideas about handling guarded expression can be easily transferred to other situations. Using formulas as guards provides a powerful tool for heuristically reducing the combinatorial explosion of cases: equivalent, redundant, tautological, and contradictive cases can be detected by simplification and quantifier elimination. Our approach allows to simplify the expressions on the basis of simplification knowledge on the logical side. The method described in this paper is implemented in the REDUCE package GUARDIAN, which is freely available on the WWW.

1 Introduction

It is meanwhile a well-known fact that evaluations obtained with the interactive use of computer algebra systems (CAS) are not entirely correct in general. Typically, some degenerate cases are dropped. Consider for instance the evaluation

$$\frac{x^2}{x} = x,$$

which is correct only if $x \neq 0$. The problem here is that CAS consider variables to be transcendental elements. The user, in contrast, has in mind variables in the sense of logic. In other words: The user does not think of rational functions but of terms.

Next consider the valid expression

$$\frac{\sqrt{x} + \sqrt{-x}}{x}.$$

It is meaningless over the reals. CAS often offer no choice than to interprete surds over the complex numbers even if they distinguish between a *real* and a *complex* mode.

Corless and Jeffrey [CJ92] have examined the behavior of a number of CAS with such input data. They come to the conclusion that simultaneous computation of all cases is exemplary but not feasible due to the combinatorial explosion of cases to be considered. Therefore, they suggest to ignore the degenerate cases but to provide the assumptions to the user on request. We claim, in contrast, that it is in fact feasible to compute all possible cases.

Our setting is as follows: Expressions are evaluated to *guarded expressions* consisting of possibly several conventional expressions guarded by quantifier-free formulas. For the above examples, we would obtain

$$\left[\ \boldsymbol{x \neq 0} \ \middle|\ \boldsymbol{x}\ \right], \quad \left[\ \mathbf{F} \ \middle|\ \boldsymbol{\tfrac{\sqrt{x}+\sqrt{-x}}{x}}\ \right].$$

As the second example illustrates, we are working in ordered fields, more precisely in real closed fields. The handling of guarded expressions as described in this paper can, however, be easily transferred to other situations.

Our approach can also deal with redundant guarded expressions, such as

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{|x| - x} \\ x \geq 0 & 0 \\ x < 0 & -2x \end{array}\right]$$

which leads to algebraic simplification techniques based on logical simplification as proposed by Davenport and Faure [DF94].

We use *formulas* over the language of ordered rings as guards. This provides powerful tools for heuristically reducing the combinatorial explosion of cases: equivalent, redundant, tautological, and contradictive cases can be detected by *simplification* [DS95] and *quantifier elimination* [Tar48, Col75, Wei88, LW93, Wei97, Wei94]. In certain situations, we will allow the formulas also to contain extra functions such as $\sqrt{\cdot}$ or $|\cdot|$. Then we take care that there is no quantifier elimination applied.

Simultaneous computation of several cases concerning certain expressions being zero or not has been extensively investigated as *dynamic evaluation* [GD93, DR94a, DR94b, BGDW95]. It has also been extended to real closed fields [DGV93]. The idea behind the development of these methods is of a more theoretical nature than to overcome the problems with the interactive usage of CAS sketched above: one wishes to compute in algebraic (or real) extension fields of the rationals. Guarded expressions occur naturally when solving problems parametrically. Consider, e.g., the *Gröbner systems* used during the computation of *comprehensive Gröbner bases* [Wei92].

The algorithms described in this paper are implemented in the REDUCE package GUARDIAN. It is based on the REDUCE [HF95, Mel95] package REDLOG [DS96a, DS96b] implementing a formula data type with corresponding algorithms, in particular including simplification and quantifier elimination. Both GUARDIAN and REDLOG are available on the WWW.[1]

2 An outline of our method

2.1 Guarded expressions

A *guarded expression* is a scheme

$$\left[\begin{array}{c|c} \boldsymbol{\gamma_0} & \boldsymbol{t_0} \\ \gamma_1 & t_1 \\ \vdots & \vdots \\ \gamma_n & t_n \end{array}\right]$$

where each γ_i is a quantifier-free formula, the *guard*, and each t_i is an associated *conventional expression.* The idea is that some t_i is a valid interpretation iff γ_i holds. Each pair (γ_i, t_i) is called a *case.*

The first case (γ_0, t_0) is the *generic* case: t_0 is the expression the system would compute without our package, and γ_0 is the corresponding guard.

The guards γ_i need neither exclude one another, nor do we require that they form a complete case distinction. We shall, however, assume that all

[1] `http://www.fmi.uni-passau.de/~redlog/`

cases covered by a guarded expression are already covered by the generic case; in other words:

$$\bigwedge_{i=1}^{n} (\gamma_i \longrightarrow \gamma_0). \tag{1}$$

Consider the following evaluation of $|x|$ to a guarded expression:

$$\left[\begin{array}{c|c} \mathbf{T} & |\boldsymbol{x}| \\ x \geq 0 & x \\ x < 0 & -x \end{array}\right].$$

Here the non-generic cases already cover the whole domain. The generic case is in some way *redundant.* It is just present for keeping track of the system's default behavior. Formally we have

$$\Big(\bigvee_{i=1}^{n} \gamma_i\Big) \longleftrightarrow \gamma_0. \tag{2}$$

As an example for a non-redundant, i.e., *necessary* generic case we have the evaluation of the reciprocal $\frac{1}{x}$:

$$\left[\begin{array}{c|c} \boldsymbol{x \neq 0} & \boldsymbol{\frac{1}{x}} \end{array}\right].$$

In every guarded expression, the generic case is explicitly marked as either necessary or redundant. The corresponding tag is inherited during the evaluation process. Unfortunately it can happen that guarded expressions satisfy (2) without being tagged redundant, e.g., specialization of

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{\sin x} \\ x = 0 & 0 \end{array}\right]$$

to $x = 0$ if the system cannot evaluate $\sin(0)$. This does not happen if one claims for necessary generic cases to have, as the reciprocal above, no alternative cases at all. Else, in the sequel "redundant generic case" has to be read as "tagged redundant."

With guarded expressions, the evaluation splits into two independent parts: *Algebraic evaluation* and a subsequent *simplification* of the guarded expression obtained.

2.2 Guarding schemes

In the introduction we have seen that certain operators introduce case distinctions. For this, with each operator f there is a *guarding scheme* associated providing information on how to map $f(t_1, \ldots, t_m)$ to a guarded expression provided that one does not have to care for the argument expressions $t_1, \ldots, t_m$. In the easiest case, this is a rewrite rule

$$f(a_1, \ldots, a_m) \to G(a_1, \ldots, a_m).$$

The actual terms $t_1, \ldots, t_m$ are simply substituted for the formal symbols $a_1, \ldots, a_m$ into the generic guarded expression $G(a_1, \ldots, a_m)$. We give some examples:

$$\begin{aligned}
\frac{a_1}{a_2} &\to \left[\begin{array}{c|c} \boldsymbol{a_2 \neq 0} & \boldsymbol{\frac{a_1}{a_2}} \end{array}\right] \\
\sqrt{a_1} &\to \left[\begin{array}{c|c} \boldsymbol{a_1 \geq 0} & \boldsymbol{\sqrt{a_1}} \end{array}\right] \\
\operatorname{sign}(a_1) &\to \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{\mathrm{sign}(a_1)} \\ a_1 > 0 & 1 \\ a_1 = 0 & 0 \\ a_1 < 0 & -1 \end{array}\right] \\
|a_1| &\to \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{|a_1|} \\ a_1 \geq 0 & a_1 \\ a_1 < 0 & -a_1 \end{array}\right]
\end{aligned} \tag{3}$$

For functions of arbitrary arity, e.g., min or max, we formally assume infinitely many operators of the same name. Technically, we associate a procedure parameterized with the number of arguments m that generates the corresponding rewrite rule. As `min_scheme`(2) we obtain, e.g.,

$$\min(a_1, a_2) \to \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{\min(a_1, a_2)} \\ a_1 \leq a_2 & a_1 \\ a_2 \leq a_1 & a_2 \end{array}\right], \tag{4}$$

while for higher arities there are more case distinctions necessary.

For later complexity analysis, we state the concept of a guarding scheme formally: a guarding scheme for an m-ary operator f is a map

$$\mathrm{gscheme}_f : \mathrm{E}^m \to \mathrm{GE}$$

where E is the set of expressions, and GE is the set of guarded expressions. This allows to split $f(t_1, \ldots, t_m)$ in dependence on the form of the parameter expressions $t_1, \ldots, t_m$.

2.3 Algebraic evaluation

2.3.1 Evaluating conventional expressions

The evaluation of conventional expressions into guarded expressions is performed recursively: Constants c evaluate to

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{c} \end{array}\right].$$

For the evaluation of $f(e_1, \dots, e_m)$ the argument expressions $e_1, \dots, e_m$ are recursively evaluated to guarded expressions

$$e'_i = \left[\begin{array}{c|c} \boldsymbol{\gamma_{i0}} & \boldsymbol{t_{i0}} \\ \gamma_{i1} & t_{i1} \\ \vdots & \vdots \\ \gamma_{in_i} & t_{in_i} \end{array}\right] \quad \text{for} \quad 1 \le i \le m. \tag{5}$$

Then the operator f is "moved inside" the e'_i by combining all cases, technically a simultaneous Cartesian product computation of both the sets of guards and the sets of terms:

$$\Gamma = \prod_{i=1}^{m} \{\gamma_{i0}, \dots, \gamma_{in_i}\}, \quad T = \prod_{i=1}^{m} \{t_{i0}, \dots, t_{in_i}\}. \tag{6}$$

This leads to the intermediate result

$$\left[\begin{array}{c|c} \boldsymbol{\gamma_{10} \wedge \dots \wedge \gamma_{m0}} & \boldsymbol{f(t_{10}, \dots, t_{m0})} \\ \vdots & \vdots \\ \gamma_{1n_1} \wedge \dots \wedge \gamma_{m0} & f(t_{1n_1}, \dots, t_{m0}) \\ \vdots & \vdots \\ \gamma_{1n_1} \wedge \dots \wedge \gamma_{mn_m} & f(t_{1n_1}, \dots, t_{mn_m}) \end{array}\right]. \tag{7}$$

The new generic case is exactly the combination of the generic cases of the e'_i. It is redundant if at least one of these combined cases is redundant.

Next, all non-generic cases containing at least one *redundant* generic constituent γ_{i0} in their guard are deleted. The reason for this is that generic cases are only used to keep track of the system default behavior. All other cases get the status of a non-generic case even if they contain necessary generic constituents in their guard.

At this point, we apply the guarding scheme of f to all remaining expressions $f(t_{1i_1}, \dots, t_{mi_m})$ in the form (7) yielding a nested guarded expression

$$\left[\begin{array}{c|c} \boldsymbol{\Gamma_0} & \left[\begin{array}{c|c} \boldsymbol{\delta_{00}} & \boldsymbol{u_{00}} \\ \vdots & \vdots \\ \delta_{0k_0} & u_{0k_0} \end{array}\right] \\ \vdots & \vdots \\ \Gamma_N & \left[\begin{array}{c|c} \boldsymbol{\delta_{N0}} & \boldsymbol{u_{N0}} \\ \vdots & \vdots \\ \delta_{Nk_N} & u_{Nk_N} \end{array}\right] \end{array}\right], \tag{8}$$

which can be straightforwardly resolved to a guarded expression

$$\left[\begin{array}{c|c} \boldsymbol{\Gamma_0 \wedge \delta_{00}} & \boldsymbol{u_{00}} \\ \vdots & \vdots \\ \Gamma_0 \wedge \delta_{0k_0} & u_{0k_0} \\ \vdots & \vdots \\ \Gamma_N \wedge \delta_{N0} & u_{N0} \\ \vdots & \vdots \\ \Gamma_N \wedge \delta_{Nk_N} & u_{Nk_N} \end{array}\right].$$

This form is treated analogously to the form (7): The new generic case $(\Gamma_0 \wedge \delta_{00}, u_{00})$ is redundant if at least one of $(\Gamma_0, f(t_{10}, \dots, t_{m0}))$ and (δ_{00}, u_{00}) is redundant. Among the non-generic cases all those containing redundant generic constituents in their guard are deleted, and all those containing necessary generic constituents in their guard get the status of an ordinary non-generic case.

Finally the standard evaluator of the system—`reval` in the case of REDUCE—is applied to all contained expressions, which completes the algebraic part of the evaluation.

2.3.2 Evaluating guarded expressions

The previous section was concerned with the evaluation of pure conventional expressions into guarded expressions. Our system currently combines both conventional and guarded expressions. We are thus faced with the problem of treating guarded subexpressions during evaluation.

When there is a *guarded* subexpression e_i detected during evaluation, all contained expressions are recursively evaluated to guarded expressions yielding a nested guarded expression of the form (8). This is resolved as described above yielding the evaluation subresult e_i'.

As a special case, this explains how guarded expressions are (re)evaluated to guarded expressions.

2.4 Example

We describe the evaluation of the expression $\min(x, |x|)$. The first argument $e_1 = x$ evaluates recursively to

$$e_1' = \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{x} \end{array}\right] \tag{9}$$

with a necessary generic case. The nested x inside $e_2 = |x|$ evaluates to the same form (9). For obtaining e_2', we apply the guarding scheme (3) of the absolute value to the only term of (9) yielding

$$\left[\begin{array}{c|c} \mathbf{T} & \left[\begin{array}{c|c} \mathbf{T} & |\boldsymbol{x}| \\ x \geq 0 & x \\ x < 0 & -x \end{array}\right] \end{array}\right],$$

where the inner generic case is redundant. This form is resolved to

$$e_2' = \left[\begin{array}{c|c} \mathbf{T} \wedge \mathbf{T} & |\boldsymbol{x}| \\ \mathrm{T} \wedge x \geq 0 & x \\ \mathrm{T} \wedge x < 0 & -x \end{array}\right]$$

with a redundant generic case. The next step is the combination of cases by Cartesian product computation. We obtain

$$\left[\begin{array}{c|c} \mathbf{T} \wedge (\mathbf{T} \wedge \mathbf{T}) & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ \mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0) & \min(x, x) \\ \mathrm{T} \wedge (\mathrm{T} \wedge x < 0) & \min(x, -x) \end{array}\right],$$

which corresponds to (7) above. For the outer min, we apply the guarding scheme (4) to all terms yielding the nested guarded expression

$$\left[\begin{array}{c|c} \mathbf{T} \wedge (\mathbf{T} \wedge \mathbf{T}) & \left[\begin{array}{c|c} \mathbf{T} & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ x \leq |x| & x \\ |x| \leq x & |x| \end{array}\right] \\ \mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0) & \left[\begin{array}{c|c} \mathbf{T} & \mathbf{min}(\boldsymbol{x}, \boldsymbol{x}) \\ x \leq x & x \\ x \leq x & x \end{array}\right] \\ \mathrm{T} \wedge (\mathrm{T} \wedge x < 0) & \left[\begin{array}{c|c} \mathbf{T} & \mathbf{min}(\boldsymbol{x}, -\boldsymbol{x}) \\ x \leq -x & x \\ -x \leq x & -x \end{array}\right] \end{array}\right],$$

which is in turn resolved to

$$\left[\begin{array}{c|c} (\mathbf{T} \wedge (\mathbf{T} \wedge \mathbf{T})) \wedge \mathbf{T} & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ (\mathrm{T} \wedge (\mathrm{T} \wedge \mathrm{T})) \wedge x \leq |x| & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge \mathrm{T})) \wedge |x| \leq x & |x| \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0)) \wedge \mathrm{T} & \min(x, x) \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0)) \wedge x \leq x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0)) \wedge x \leq x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x < 0)) \wedge \mathrm{T} & \min(x, -x) \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x < 0)) \wedge x \leq -x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x < 0)) \wedge -x \leq x & -x \end{array}\right].$$

From this, we delete the two non-generic cases obtained by combination with the redundant generic case of the min. The final result of the algebraic evaluation step is the following:

$$\left[\begin{array}{c|c} \mathbf{(T \wedge (T \wedge T)) \wedge T} & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ (\mathrm{T} \wedge (\mathrm{T} \wedge \mathrm{T})) \wedge x \leq |x| & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge \mathrm{T})) \wedge |x| \leq x & |x| \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0)) \wedge x \leq x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x \geq 0)) \wedge x \leq x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x < 0)) \wedge x \leq -x & x \\ (\mathrm{T} \wedge (\mathrm{T} \wedge x < 0)) \wedge -x \leq x & -x \end{array}\right]. \tag{10}$$

2.5 Worst-case complexity

Our measure of complexity $|G|$ for guarded expressions G is the number of contained cases:

$$\left\|\left[\begin{array}{c|c} \boldsymbol{\gamma_0} & \boldsymbol{t_0} \\ \gamma_1 & t_1 \\ \vdots & \vdots \\ \gamma_n & t_n \end{array}\right]\right\| = n+1.$$

As in Section 2.3, consider an m-ary operator f, guarded expression arguments $e_1', \dots, e_m'$ as in equation (5), and the Cartesian product T as in equation (6). Then

$$\begin{aligned} |f(e_1', \dots, e_m')| &\leq \sum_{(t_1,\dots,t_m)\in T} |\mathrm{gscheme}_f(t_1, \dots, t_m)| \\ &\leq \max_{(t_1,\dots,t_m)\in T} |\mathrm{gscheme}_f(t_1, \dots, t_m)| \cdot \#T \\ &= \max_{(t_1,\dots,t_m)\in T} |\mathrm{gscheme}_f(t_1, \dots, t_m)| \cdot \prod_{j=1}^{m} |e_j'| \\ &\leq \max_{(t_1,\dots,t_m)\in T} |\mathrm{gscheme}_f(t_1, \dots, t_m)| \cdot (\max_{1\leq j\leq m} |e_j'|)^m. \end{aligned}$$

In the important special case that the guarding scheme of f is a rewrite rule $f(a_1, \dots, a_m) \to G$, the above complexity estimation simplifies to

$$|f(e_1', \dots, e_m')| \leq |G| \cdot \prod_{j=1}^{m} |e_j'| \leq |G| \cdot (\max_{1\leq j\leq m} |e_j'|)^m.$$

In other words: $|G|$ plays the role of a factor, which, however, depends on f, and $|f(e_1', \dots, e_m')|$ is polynomial in the size of the e_i but exponential in the arity of f.

2.6 Simplification

In view of the increasing size of the guarded expressions coming into existence with subsequent computations, it is indispensable to apply simplification strategies. There are two different algorithms involved in the simplification of guarded expressions:

1. A *formula simplifier* mapping quantifier-free formulas to equivalent simpler ones.

2. Effective *quantifier elimination* for real closed fields over the language of ordered rings.

It is not relevant, which simplifier and which quantifier elimination procedure is actually used. We use the formula simplifier described in [DS95]. Our quantifier elimination uses test point methods developed by Weispfenning [Wei88, LW93, Wei97]. It is restricted to formulas obeying certain degree restrictions wrt. the quantified variables. As an alternative, REDLOG provides an interface to Hong's QEPCAD quantifier elimination package [HCJE93]. Compared to the simplification, the quantifier elimination is more time consuming. It can be turned off by a *switch*.

The following simplification steps are applied in the given order:

Contraction of cases This is restricted to the non-generic cases of the considered guarded expression. We contract different cases containing the same terms:

$$\left[\begin{array}{c|c} \boldsymbol{\gamma_0} & \boldsymbol{t_0} \\ \vdots & \vdots \\ \gamma_i & t_i \\ \vdots & \vdots \\ \gamma_j & t_i \\ \vdots & \vdots \end{array}\right] \quad \text{becomes} \quad \left[\begin{array}{c|c} \boldsymbol{\gamma_0} & \boldsymbol{t_0} \\ \vdots & \vdots \\ \gamma_i \vee \gamma_j & t_i \\ \vdots & \vdots \end{array}\right].$$

Simplification of the guards The simplifier is applied to all guards replacing them by simplified equivalents. Since our simplifier maps $\gamma \vee \gamma$ to γ, this together with the contraction of cases takes care for the deletion of duplicate cases.

Keep one tautological case If the guard of some non-generic case becomes "T," we delete all other non-generic cases. Else, if quantifier elimination is turned on, we try to detect a tautology by eliminating the universal closures $\underline{\forall}\gamma$ of the guards γ. This quantifier elimination is also applied to the guards of generic cases. These are, in case of success, simply replaced by "T" without deleting the case.

Remove contradictive cases A non-generic case is deleted if its guard has become "F." If quantifier elimination is turned on, we try to detect further contradictive cases by eliminating the existential closure $\exists\gamma$ for each guard γ. This quantifier elimination is also applied to generic cases. In case of success they are not deleted but their guards are replaced by "F." Our assumption (1) allows then to delete all non-generic cases.

2.7 Example revisited

We turn back to the form (10) of our example $\min(x, |x|)$. Contraction of cases with subsequent simplification automatically yields

$$\left[\begin{array}{c|c} \mathbf{T} & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ \mathrm{T} & x \\ |x| - x \leq 0 & |x| \\ \mathrm{F} & -x \end{array}\right],$$

of which only the tautological non-generic case survives:

$$\left[\begin{array}{c|c} \mathbf{T} & \mathbf{min}(\boldsymbol{x}, |\boldsymbol{x}|) \\ \mathrm{T} & x \end{array}\right]. \tag{11}$$

2.8 Output modes

An *output mode* determines which part of the information contained in the guarded expressions is provided to the user. GUARDIAN knows the following output modes:

Matrix Output matrices in the style used throughout this paper. We have already seen that these can become very large in general.

Generic case Output only the generic case.

Generic term Output only the generic term. Thus the output is exactly the same as without the guardian package. If the condition of the generic case becomes "F," a *warning* "`contradictive situation`" is given. The computation can, however, be continued.

Note that output modes are restrictions concerning only the output; internally the system still computes with the complete guarded expressions.

2.9 A smart mode

Consider the evaluation result (11) of $\min(x, |x|)$. The *generic term* output mode would output $\min(x, |x|)$, although more precise information could be given, namely x. The problem is caused by the fact that generic cases are used to keep track of the system's default behavior. In this section we will describe an optional *smart mode* with a different notion of *generic case*. To begin with, we show why the problem can not be overcome by a "smart output mode."

Assume that there is an output mode which outputs x for (11). As the next computation involving (11) consider division by y. This would result in

$$\left[\begin{array}{c|c} \boldsymbol{y \neq 0} & \boldsymbol{\frac{\min(x,|x|)}{y}} \\ y \neq 0 & \frac{x}{y} \end{array}\right].$$

Again, there are identic conditions for the generic case and some non-generic case, and, again, the term belonging to the latter is simpler. Our mode would output $\frac{x}{y}$. Next, we apply the absolute value once more yielding

$$\left[\begin{array}{c|c} \boldsymbol{y \neq 0} & \boldsymbol{\frac{|\min(x,|x|)|}{|y|}} \\ xy \geq 0 \land y \neq 0 & \frac{x}{y} \\ xy < 0 \land y \neq 0 & \frac{-x}{y} \end{array}\right].$$

Here, the condition of the generic case differs from all other conditions. We thus have to output the generic term. For the user, the evaluation of $|\frac{x}{y}|$ results in $\frac{|\min(x,|x|)|}{|y|}$.

The smart mode can turn a non-generic case into a necessary generic one dropping the original generic case and all other non-generic cases. Consider, e.g., (11), where the conditions are equal, and the non-generic term is "simpler."

In fact, the relevant relationship between the conditions is that the generic condition *implies* the non-generic one. In other words: Some non-generic condition is not more restrictive than the generic condition, and thus covers the whole domain of the guarded expression. Note that from the implication and (1) we may conclude that the cases are even equivalent.

Implication is heuristically checked by simplification. If this fails, quantifier elimination provides a decision procedure. Note that our test point methods are incomplete in this regard due to the degree restrictions. Also it cannot be applied straightforwardly to guards containing operators that do not belong to the language of ordered rings.

Whenever we happen to detect a relevant implication, we actually turn the corresponding non-generic case into the generic one. From our motivation of non-generic cases, we may expect that non-generic expressions are generally more convenient than generic ones.

3 Examples

We give the results for the following computations as they are printed in the output mode *matrix* providing the full information on the computation result. The reader can derive himself what the output in the mode *generic case* or *generic term* would be.

- Smart mode or not:
$$\frac{1}{x^2+2x+1} = \left[\ \boldsymbol{x+1 \neq 0} \ \middle|\ \boldsymbol{\frac{1}{x^2+2x+1}}\ \right].$$
The simplifier recognizes that the denominator is a square.

- Smart mode or not:
$$\frac{1}{x^2+2x+2} = \left[\ \mathbf{T} \ \middle|\ \boldsymbol{\frac{1}{x^2+2x+2}}\ \right].$$
Quantifier elimination recognizes the positive definiteness of the denominator.

- Smart mode:
$$|x| - \sqrt{x} = \left[\ \boldsymbol{x \geq 0} \ \middle|\ \boldsymbol{-\sqrt{x}+x}\ \right].$$
The square root allows to forget about the negative branch of the absolute value.

- Smart mode:
$$|x^2+2x+1| = \left[\ \mathbf{T} \ \middle|\ \boldsymbol{x^2+2x+1}\ \right].$$
The simplifier recognizes the positive semidefiniteness of the argument. Reduce itself recognizes squares within absolute values only in very special cases such as $|x^2|$.

- Smart mode:
$$\min(x, \max(x,y)) = \left[\ \mathbf{T} \ \middle|\ \boldsymbol{x}\ \right].$$
Note that REDUCE does not know any rules about nested minima and maxima.

- Smart mode:
$$\min(\mathrm{sign}(x), -1) = \left[\ \mathbf{T} \ \middle|\ \boldsymbol{-1}\ \right].$$

- Smart mode or not:

$$|x| - x = \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{|x| - x} \\ x \geq 0 & 0 \\ x < 0 & -2x \end{array}\right].$$

  This example is taken from [DF94].

- Smart mode or not:

$$\sqrt{1 + x^2y^2(x^2 + y^2 - 3)} = \left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{\sqrt{x^4y^2 + x^2y^4 - 3x^2y^2 + 1}} \end{array}\right]$$

  The *Motzkin polynomial* is recognized to be positive semidefinite by quantifier elimination.

The evaluation time for the last example is 119 ms on a SUN SPARC-4. This illustrates that efficiency is no problem with such interactive examples.

4 Outlook

This section describes possible extensions of the GUARDIAN. The extensions proposed in Section 4.1 on simplification of terms and Section 4.2 on a background theory are clear from a theoretical point of view but not yet implemented. Section 4.3 collects some ideas on the application of our ideas to the REDUCE integrator. In this field, there is some more theoretical work necessary.

4.1 Simplification of terms

Consider the expression $\mathrm{sign}(x)x - |x|$. It evaluates to the following guarded expression:

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{-|x| + \mathrm{sign}(x)x} \\ x \neq 0 & 0 \\ x = 0 & -x \end{array}\right].$$

This suggests to substitute $-x$ by 0 in the third case, which would in turn allow to contract the two non-generic cases yielding

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{-|x| + \mathrm{sign}(x)x} \\ \mathrm{T} & 0 \end{array}\right].$$

In smart mode second case would then become the only generic case.

Generally, one would proceed as follows: If the guard is a conjunction containing as toplevel equations

$$t_1 = 0, \quad \ldots, \quad t_k = 0,$$

reduce the corresponding expression modulo the set of univariate linear polynomials among $t_1, \ldots, t_k$.

A more general approach would reduce the expression modulo a Gröbner basis of all the $t_1, \ldots, t_k$. This leads, however, to larger expressions in general.

One can also imagine to make use of non-conjunctive guards in the following way:

1. Compute a DNF of the guard.
2. Split the case into several cases corresponding to the conjunctions in the DNF.
3. Simplify the terms.
4. Apply the standard simplification procedure to the resulting guarded expression. Note that it includes *contraction of cases*.

According to experiences with similar ideas in the "Gröbner simplifier" described in [DS95], this should work well.

4.2 Background theory

In practice one often computes with quantities guaranteed to lie in a certain range. For instance, when computing an electrical resistance, one knows in advance that it will not be negative. For such cases one would like to have some facility to provide external information to the system. This can then be used to reduce the complexity of the guarded expressions.

One would provide a function `assert`(φ), which asserts the formula φ to hold. Successive applications of `assert` establish a *background theory*, which is a set of formulas considered conjunctively. The information contained in the background theory can be used with the guarded expression computation. The user must, however, not rely on all the background information to be actually used.

Technically, denote by Φ the (conjunctive) background theory. For the *simplification of the guards*, we can make use of the fact that our simplifier is designed to simplify wrt. a theory, cf. [DS95]. For proving that some guard γ is *tautological*, we try to prove

$$\underline{\forall}(\Phi \longrightarrow \gamma)$$

instead of $\underline{\forall}\gamma$. Similarly, for proving that γ is *contradictive*, we try to disprove

$$\underline{\exists}(\Phi \wedge \gamma).$$

Instead of proving $\underline{\forall}(\gamma_1 \longrightarrow \gamma_2)$ in smart mode, we try to prove

$$\underline{\forall}((\Phi \wedge \gamma_1) \longrightarrow \gamma_2).$$

Independently, one can imagine to use a background theory for reducing the *output* with the *matrix* output mode. For this, one simplifies each guard wrt. the theory at the output stage treating contradictions and tautologies appropriately. Using the theory for replacing all cases by one at output stage in a smart mode manner leads once more to the problem of expressions or even guarded expressions "mysteriously" getting more complicated. Applying the theory only at the output stage makes it possible to implement a procedure `unassert(`φ`)` in a reasonable way.

4.3 Integration

CAS integrators make "mistakes" similar to those we have examined. Consider, e.g., the typical result

$$\int x^a\,dx = \frac{1}{a+1}x^{a+1}.$$

It does not cover the case $a = -1$, for which one wishes to obtain

$$\int x^{-1}\,dx = \ln x.$$

This problem can also be solved by using guarded expressions for integration results.

Within the framework of this paper, we would have to associate a guarding scheme to the integrator `int`. It is not hard to see that this cannot be done in a reasonable way without putting as much knowledge into the scheme as into the integrator itself. Thus for treating integration, one has to modify the integrator to provide guarded expressions.

Next, we have to clarify what the guarded expression for the above integral would look like. Since we know that the integral is defined for all interpretations of the variables, our assumption (1) implies that the generic condition be "T." We obtain the guarded expression

$$\left[\begin{array}{c|c} \mathbf{T} & \boldsymbol{\int x^a\,dx} \\ a \neq -1 & \frac{1}{a+1}x^{a+1} \\ a = -1 & \ln x \end{array}\right].$$

Note that the redundant generic case does not model the system's current behavior.

4.4 Combining algebra with logic

Our method, in the described form, uses an already implemented algebraic evaluator. In the previous section, we have seen that this point of view is not sufficient for treating integration appropriately.

Also our approach runs into trouble with built-in knowledge such as

$$\begin{aligned} \sqrt{x^2} &= |x|, & (12) \\ \operatorname{sign}(|x|) &= 1. & (13) \end{aligned}$$

Equation (12) introduces an absolute value operator within a non-generic term without making a case distinction. Equation (13) is wrong when not considering x transcendental. In contrast to the situation with reciprocals, our technique cannot be used to avoid this "mistake." We obtain

$$\operatorname{sign}(|x|) = \left[\begin{array}{c|c} \mathbf{T} & \mathbf{1} \\ x \neq 0 & 1 \\ x = 0 & 0 \end{array}\right]$$

yielding two different answers for $x = 0$.

We have already seen in the example Section 3 that the implementation of knowledge such as (12) and (13) is usually quite *ad hoc*, and can be mostly covered by using guarded expressions. This obesrvation gives rise to the following question: When designing a new CAS based on guarded expressions, how should the knowledge be distributed between the algebraic side and the logic side?

5 Conclusions

Guarded expressions can be used to overcome well-known problems with interpreting expressions as terms. We have explained in detail how to compute with guarded expressions including several simplification techniques. Moreover we gain algebraic simplification power from the logical simplifications. Numerous examples illustrate the power of our simplification methods. The largest part of our ideas is efficiently implemented, and the software is published. The outlook on background theories and on the treatment of integration by guarded expressions points on interesting future extensions.

References

[BGDW95] P.A. Broadberry, Teresa Gómez-Díaz, and S.M. Watt. On the implementation of dynamic evaluation. In A.H.M. Levelt, editor, *Proceedings of the International Symposium on Symbolic* *and Algebraic Manipulation (ISSAC 95)*, pages 77–89, New York, N.Y., 1995. ACM Press.

[Bra92] Russel Bradford. Algebraic simplification of multiple valued functions. In John Fitch, editor, *Design and Implementation of Symbolic Computation Systems*, volume 721 of *Lecture Notes in Computer Science*, pages 13–21. Springer-Verlag, 1992. Proceedings of the DISCO 92.

[CJ92] Robert M. Corless and David J. Jeffrey. Well ...it isn't quite that simple. *ACM SIGSAM Bulletin*, 26(3):2–6, August 1992. Feature.

[Col75] George Edwin Collins. Quantifier elimination for the elementary theory of real closed fields by cylindrical algebraic decomposition. In H. Brakhage, editor, *Automata Theory and Formal Languages. 2nd GI Conference*, volume 33 of *Lecture Notes in Computer Science*, pages 134–183, Berlin, Heidelberg, New York, May 1975. Gesellschaft für Informatik, Springer-Verlag.

[DF94] James H. Davenport and Christèle Faure. The "unknown" in computer algebra. *Programmirovanie*, 1(1), 1994.

[DGV93] Dominique Duval and L. Gonzáles-Vega. Dynamic evaluation and real closure. In *Proceedings of the IMACS Symposium on Symbolic Computation*, 1993.

[DR94a] Dominique Duval and Jean-Claude Reynaud. Sketches and computation I: Basic definitions and static evaluation. *Mathematical Structures in Computer Science*, 4(2):185–238, 1994.

[DR94b] Dominique Duval and Jean-Claude Reynaud. Sketches and computation II: Dynamic evaluation and applications. *Mathematical Structures in Computer Science*, 4(2):239–271, 1994.

[DS95] Andreas Dolzmann and Thomas Sturm. Simplification of quantifier-free formulas over ordered fields. Technical Report MIP-9517, FMI, Universität Passau, D-94030 Passau, Germany, October 1995. To appear in the Journal of Symbolic Computation.

[DS96a] Andreas Dolzmann and Thomas Sturm. Redlog—computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universität Passau, D-94030 Passau, Germany, February 1996.

[DS96b] Andreas Dolzmann and Thomas Sturm. Redlog user manual. Technical Report MIP-9616, FMI, Universität Passau, D-94030 Passau, Germany, October 1996. Edition 1.0 for Version 1.0.

[GD93] Teresa Gómez-Díaz. Examples of using dynamic constructible closure. In *Proceedings of the IMACS Symposium on Symbolic Computation*, 1993.

[HCJE93] Hoon Hong, George E. Collins, Jeremy R. Johnson, and Mark J. Encarnacion. QEPCAD interactive version 12. Kindly communicated to us by Hoon Hong, September 1993.

[HF95] Anthony C. Hearn and John P. Fitch. *Reduce User's Manual for Version 3.6.* RAND, Santa Monica, CA 90407-2138, July 1995. RAND Publication CP78.

[LW93] Rüdiger Loos and Volker Weispfenning. Applying linear quantifier elimination. *The Computer Journal*, 36(5):450–462, 1993. Special issue on computational quantifier elimination.

[Mel95] Herbert Melenk. Reduce symbolic mode primer. In *REDUCE 3.6 User's Guide for UNIX*. Konrad-Zuse-Institut, Berlin, 1995.

[Tar48] A. Tarski. A decision method for elementary algebra and geometry. Technical report, University of California, 1948. Second edn., rev. 1951.

[Wei88] Volker Weispfenning. The complexity of linear problems in fields. *Journal of Symbolic Computation*, 5(1):3–27, February 1988.

[Wei92] Volker Weispfenning. Comprehensive Gröbner bases. *Journal of Symbolic Computation*, 14:1–29, July 1992.

[Wei94] Volker Weispfenning. Quantifier elimination for real algebra—the cubic case. In *Proceedings of the International Symposium on Symbolic and Algebraic Computation in Oxford*, pages 258–263, New York, July 1994. ACM Press.

[Wei97] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. 1997. To appear in AAECC.