Research Memorandum ISIS-RM-5E

Real Quadratic Quantifier Elimination in Risa/Asir

Thomas Sturm*

September 27, 1996

Institute for Social Information Science (*ISIS*),
FUJITSU LABORATORIES LIMITED.

Numazu office
140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
Telephone: (Numazu) 0559-23-2222 Telex: 3922508J Fax: 0559-24-6180

Tokyo office
1-9-3, Nakase, Mihama-ku, Chiba-shi, Chiba 261, Japan
Telephone: (Chiba) 043-299-3211 Fax: 043-299-3075

Real Quadratic Quantifier Elimination in Risa/Asir

Thomas Sturm*

Institute for Social Information Science (*ISIS*),
FUJITSU LABORATORIES LIMITED.

140 Miyamoto, Numazu-shi, Shizuoka 410-03, Japan
Email: `sturm@iias.flab.fujitsu.co.jp`

Abstract

Weispfenning has shown how to use test term methods for quantifier elimination in linear and quadratic first-order formulas over real closed fields. This paper describes the state of the implementation of such methods in the computer algebra system Risa/Asir. The package described here is entirely written in the C programming language. We point on possible extensions of the package and give examples for automatic quantifier eliminations performed by Risa/Asir.

Key words: real quantifier elimination, parametric constraint solving, implementation, Asir package

*Visiting researcher. Original affiliation: Fakultät für Mathematik und Informatik, Universität Passau, D-94030 Passau, Germany. Email: sturm@fmi.uni-passau.de.

1 Introduction

A *quantifier elimination procedure* is an algorithm that, given a first order formula, computes an equivalent quantifier-free formula. A quantifier elimination procedure for the reals has been given already by Tarski [Tar48]. Due to the enormous power of such algorithms efforts to implement them have been made from the very beginning on. In fact the US RAND Corporation had soon tried to implement the original Tarski procedure, which certainly failed at those days.

The first complete implementation by Arnon was finished in 1981 [Arn81]. It used the cylindrical algebraic decomposition method by Collins [Col75]. This implementation of quantifier elimination triggered the development of one of the first computer algebra systems: SAC, now SAC-II/ALDES. Important improvements of the CAD method have been made by Collins and Hong [CH91] resulting in the *partial* CAD implemented in Hong's QEPCAD package [HCJE93].

Weispfenning introduced an alternative approach to quantifier elimination first for linear formulas [Wei88]. This could be extended to arbitrary degrees [LW93, Wei96b, Wei94b]. For degrees 1 and 2, the methods have been successfully implemented in the REDUCE package REDLOG written by the author and others [DS95a, DS96]. Despite the bad theoretical complexity [Wei88, DH88] and the degree restrictions REDLOG turned out suitable for solving a number of practical problems [Wei94a, Wei96a]. Meanwhile it is used commercially as part of an error diagnosis system for physical networks.

We have now started a second implementation of the test term method in the C language using the Risa/Asir computer algebra system [NT92]. The aim of this reimplementation is twofold. On one hand, we may expect to achieve a gain in efficiency due to both the C language and the efficient polynomial algorithms present in Risa. On the other hand we wish to make the methods available to interested parties in industry where C is still the most widespread programming language.

The purpose of this report is to summarize the state of the reimplementation after an initial phase of eight weeks. We further point on features to be added. All these features are already part of the REDLOG package.

2 Description of the implementation

In the sequel, we assume the reader to be familiar with the theory of quantifier elimination by test term methods as it has been described by Weispfenning [LW93, Wei96b]. Figure 1 shows the module structure of the packege.

2.1 Functionality

The current implementation can eliminate existential and universal quantifiers from prenex first-order formulas, subject to the restriction that with the elimination of each variable the latter occurs at most quadratically within the quantified *matrix* formula. This includes the technique described in [Wei96b] where in a situation as

$$\exists x\Big(a_2x^2 + a_1x + a_0 = 0 \land \varphi\Big)$$

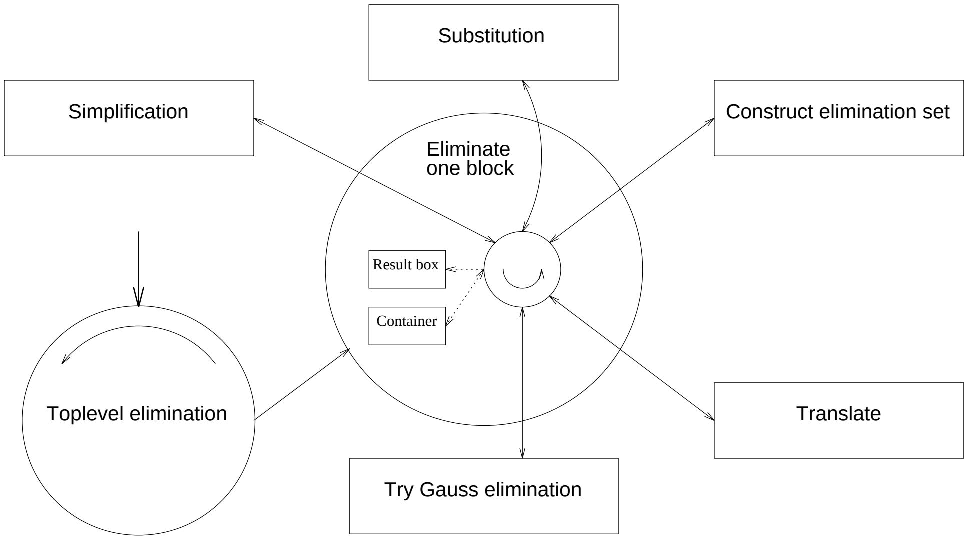

Figure 1: Visualisation of the module interaction

at least one of a_2, a_1, a_0 is a non-zero number. The quantified variable x may then occur with arbitrary degree in φ. We refer to this special case as *Gauss elimination* treated in Subsection 2.4.

Furthermore, there are some minor tool functions operating on formulas available. All functions are listed and described briefly in the Appendix A.

2.2 Toplevel elimination

The quantifier elimination routine is called with a prenex first-order formula. This formula is simplified, and then turned into *negation normal form*, i.e., a prenex formula the matrix of which contains only $\wedge$ and $\vee$ as first-order operators. Next the formula is split into a list of quantifier block and its matrix.

The elimination of universal quantifiers blocks is reduced to that of existential ones using the equivalence $\forall x\varphi \longleftrightarrow \neg\exists x\neg\varphi$. We shall restrict our attention to existential quantifiers from now on.

For each block, beginning with the innermost, there is a routine called that is designed to eliminate one prenex existential quantifier block.

2.3 Quantifier blocks

A block of quantifiers $\exists x_n \ldots \exists x_1$ is eliminated successesively beginning with the innermost quantifier. Let E_1 be an elimination set for the innermost quantifier $\exists x_1$ we obtain a formula

$$\exists x_n \ldots \exists x_2 \bigvee_{t \in E_1} \varphi[x /\!/ t].$$

The remaining quantifiers can be moved inside the disjunction. The branches obtained are eliminated independently decreasing the overall complexity from doubly to singly exponential.

This technique is implemented by processing a *queue* of *subproblems*. A subproblem is a pair (V_{ij}, ψ_{ij}) consisting of an (existentially quantified) variable list V_{ij} and a matrix formula ψ_{ij}. With the substitution of an elimination set for a certain variable we do not construct a disjuntion but instead add the results still containing quantified variables as new subproblems to a problem queue called *container*. The completely eliminated subproblems $(V_{nj} = \emptyset)$ are, in contrast, added to a *result box*.

The subproblems obtained this way form a tree by branching from each subproblem to its child problems until its leafs eventually enter the result box. The depth of this *elimination tree* is the number of variables in the initial V_{11}. Its width at each level is the sum of the sizes of the elimination sets obtained from the nodes of the parent level.

Choosing a queue for storing subproblems amounts to computing the elimination tree in a *breadth first search* (BFS) manner. The advantage of BFS is that equal sibling nodes can be detected when adding to the container. This prevents the recomputation of identical subtrees.

The procedure terminates when the container gets empty. This happens after finitely many steps according to König's Lemma. Finally a disjunction is constructed from the content of the result box.

A subproblem (V_{ij}, ψ_{ij}) is processed by first trying Gauss elimination for V_{ij} and ψ_{ij}. If this succeeds, an elimination set is obtained from the Gauss elimination routine. Else a translation is computed for the first (innermost) variable of V_{ij} and the matrix ψ_{ij}. This translation is turned into an elimination set by yet another routine. Elimination sets are substituted elementwise. Each substitution result is simplified and becomes a new subproblem with one variable less.

Notice that our container technique allows to eliminate the variables in a different order for each branch of the tree. The Gauss elimination already makes use of this feature.

2.4 Gauss elimination

The Gauss elimination routine first checks alle variables trying to find a possibility for a linear Gauss step, i.e. one in which the coefficient a_2 of x_2 is zero. Such an elimination does not increase the degree of the other variables. If this fails, it checks for a quadratic Gauss. If the Gauss routine detects a Gauss situation, it returns an elimination set to be substituted by the block elimination routine. Else it signals its failure.

2.5 Translation stage

The test term technique ist based on testing endpoints of intervals including $\pm\infty$ and possibly adding or subtracting some infinitesimal ϵ. In the translation phase all endpoint candidates are collected and classified wrt. their *bounding type* and their *infinity type*.

The bounding type encodes the relevant information on the constraint that has delivered the point. Table 1 collects the bounding types currently used. By the way of example, consider $4x - 7 \geq 0$, which contributes $7/4$ as a lower bound on x, i.e., BTGEQ. In contrast, $ax - 7 \geq 0$ does not contribute a lower bound; for $a < 0$ it is an upper bound. Thus the bounding type for $7/a$ in the latter example is BTWO.

Table 2 collects the infinity types and their relation to the bounding types. Points of

BTEQUAL	Equation: p
BTNEQ	Inequality: $\mathbb{R} \setminus \{p\}$
BTLEQ	Weak upper bound: $]-\infty, p]$
BTGEQ	Weak lower bound: $[p, \infty[$
BTLESSP	Strict upper bound: $]-\infty, p[$
BTGREATERP	Strict lower bound: $]p, \infty[$
BTWO	Weak ordering: Weak lower or upper bound
BTSO	Strict ordering: Strict lower or upper bound

Table 1: Interpretation of an endpoint p according to its bounding type.

infinity type	occurs with	interpretation
STD	BTEQUAL	p
EPS	BTSO, BTNEQ	p plus *or* minus ϵ
MEPS	BTLESSP	$p - \epsilon$
PEPS	BTGREATERP	$p + \epsilon$
PINF	—	∞
MINF	—	$-\infty$

Table 2: Interpretation of an endpoint p according to its infinity type. Infinity types are related to bounding types.

infinity type ϵ are *unspecified*, all others are *specified*. The types PINF and MINF are not used at the translation stage.

Taking a point labeled with its infinity type and adding conditions for its existence and its relevance yields a *guarded point*. Solutions $-c/b$ of linear contraints are guarded by $b \neq 0$. Solutions $(-b \pm \sqrt{b^2 - 4ac})/2a$ of quadratic contraints are guarded by $a \neq 0 \land b^2 - 4ac \geq 0$.

Finally, for a fixed variable a *translation* is an array, indexed by bounding types, of sets of guarded points obtained from the constraints in the matrix formula.

Actually, the endpoints are not stored as explicit terms but in the form of minimal polynomials. Consequently, one point can stand for several roots in case of quadratic constraint. Notice that all the roots are assigned both the same infinity type and bounding type.

On the other hand, one constraint can in fact produce several guarded points. For instance, consider the contraint $ax^2 + 5x + c \geq 0$. This yields a two-root guarded point

$$\Big(a \neq 0 \land 25 - 4ac \geq 0, (ax^2 + 5x + c, \mathrm{STD})\Big)$$

of bounding type BTWO. On the other hand, for $a = 0$ it is a hidden linear constraint. Hence we additionally obatin $(5 \neq 0, (5x + c, \mathrm{STD}))$ of bounding type BTGEQ.

The translation array is the input to the elimination set computation routine.

2.6 Elimination set computation

An elimination set is a single set of guarded points non of which is unspecified wrt. its infinity type. Given a translation there are thus three tasks to be performed.

1. Choose a small selection of guarded points for obtaining an elimination set.
2. Specify each unspecified guarded point (of infinity type EPS) in this selection to either PEPS or MEPS.
3. Add elements that are not induced by isolated contraints to the elimination set.

The current implementation decides between either taking all upper bounds or all lower bounds depending on the number of such bounds present. Accordingly, EPS is sepcified to either MEPS or PEPS, and either $(0 = 0, (0, \mathrm{PINF}))$ or $(0 = 0, (0, \mathrm{MINF}))$ is added. If there are no orderings at all, $(0 = 0, (x, \mathrm{STD}))$, i.e., "0" is added instead of an infinite value. Mind that at least one point has to be substituted for the case that none of the guards holds.

Summarizing, the bounding type provides information concerning the selection of a proper subset of the translation. The infinity type provides information concerning the substitution. The infinity type cannot always be completely determined at the translation stage.

2.7 Substitution

In an object oriented style, there is one procedure that can substitute a guarded point into a formula. The sustitution module includes substitution of infinite and infinitesimal symbols and substitution of root expressions within the language of ordered rings. The methods used are that described in [Wei96b]. We make use of our minimal polynomial representation for the roots by performing pseudo divison before symbolically substituting.

2.8 Simplification

Currently, the following simplification strategies are applied:

- Evaluation of variable-free atomic formulas.
- Replacing equation and inequality right hand sides by their squarefree part. Corresponding treatment of ordering atomic formulas: Here factors of even multiplicity have to enter squared.
- Removing `true` from conjunctions, `false` from disjunctions. Conjunctions containing `false` and disjunctions containing `true` are replaced by the respective truth value. Truth values are treated appropriately also with all other first-order operators.
- Removing equal subformulas from conjunctions and disjunctions. Atomic subformulas are ordered canonically and placed before their complex siblings. The order of atomic formulas is first wrt. their left hand side polynomial, and then wrt. the order of relations as in Table 3.

3 Possible Extensions

We point on possible extension of the Asir quantifier elimination code. All options mentioned here are already part of the REDLOG package [DS95a, DS96]. They have been extensively tested for their relevance.

3.1 Functionality

For many practical applications, it is necessary to add an option for *extended quantifier elimination:* This variant keeps track of the test points substituted. With the elimination of an existential quantifier block, it provides instead of a disjunction $\bigvee_{i=1}^{n} \psi_i$ a set of *guarded points*

$$\left\{(\psi_1, S_1), \ldots, (\psi_n, S_n)\right\}$$

such that whenever an interpretation of the parameters satisfies some ψ_i, the original $\exists$-quantified formula holds for this interpretation, and S_i provides some sample values for the quantified variables possibly containing $\pm\epsilon$ or $\pm\infty$.

The extension of the package to *generic quantifier elimination* [DSW96] is of the same importance. There, the quantifier elimination procedure is allowed to make certain assumption of the form $t \neq 0$ for parameter term t. These assumptions support the elimination process leading to a much smaller result, which is correct under the assumptions. Of course, the list of assumptions made is also returned to the caller. It has turned out that for the majority of practical applications, the assumptions made are "harmless". They describe some degenerate cases, which are actually not relevant.

Finally, the package should not be restricted to quantifier elimination itself but also include other useful algorithms on formulas such as CNF/DNF computation or advanced simplifiers for final results.

3.2 Toplevel elimination

There should be code added for making formulas prenex such that the quantifier elimination can be called with arbitrary formulas.

3.3 Quantifier blocks

There should be an option added to use a stack instead of a queue as container. This amounts to a DFS computation of the elimination tree. The advantage is that with (wlog. existential) decision problems, one has a good chance to detect a "true" leaf early. Computation can be aborted then. For the elimination of several blocks in a decision problem it is useful to switch dynamically from BFS to DFS.

A variable selection strategy has to be added. For instance, linear variables have to be preferred because their elimination does not increase the degree of the other variables.

Some techniques for reducing the degree of the quantified variables take into account the whole formula [DSW96]. They should be applied at this stage.

3.4 Gauss elimination

The Gauss elimination code already prefers linear variables. It should check all possibilities yet more closely: among linear equations, e.g., $x = 0$ should be preferred to $ax + 1 = 0$ since the former does not introduce a guarding condition.

One should try to factorize polynomials of a degree greater than 2.

3.5 Translation stage

At the translation stage, one also should try to factorize polynomials of a degree greater than 2. The current implementation aborts with an error in such a case. Even if factorization fails, the computation should be continued. The result can be requantified, and with an (wlog. existential) decision problem, one still has the chance to find “true”.

3.6 Elimination set computation

There are numerous sophisticated techniques for constructing small elimination sets from a translation.

3.7 Simplification

Due to the doubly exponential explosion in the number of atomic fomulas, simplification plays an extremely important role with this method. Appropriate simplifications are described in detail in [DS95b].

4 Computation examples

All the following examples have been computed on a SUN Sparc-20.

4.1 The Davenport–Heintz Example

This example is taken from [CH91], p. 325. The formula

$$\exists c \forall a \forall b (a = d \land b = c) \lor (a = c \land b = 1) \longrightarrow a^2 = b))$$

is a special case of more general formula used by Davenport and Heintz [DH88] in order to show the time complexity of real quantifier elimination. It is equivalent to $d = 1 \lor d = -1$. Asir yields after 0.03 s the equivalent result

$$d^4 - 1 = 0 \land d \neq 0 \land (d = 0 \lor d \neq 0) \land (d = 0 \lor d \neq 0 \land (d^2 - 1 \neq 0 \lor d \neq 0)) \land (d^3 - d = 0 \lor d^2 - d \neq 0 \land d^2 - 1 \neq 0) \land (d^2 - 1 \neq 0 \lor d \neq 0) \land (d^2 - 1 = 0 \lor d^2 - d \neq 0 \land d^2 - 1 \neq 0)$$

The REDLOG standard simplifier can simplify this to $d \neq 0 \land (d + 1 = 0 \lor d - 1 = 0)$. This points on the necessity of more sophisticated simplification strategies.

4.2 Transportation problem

An example which should not suffer from missing simplification is the 2-dimensional planar 3×3 transportation problem taken from [LW93], pp. 459/460. The input formula

$$\exists x_{11} \exists x_{12} \exists x_{13} \exists x_{21} \exists x_{22} \exists x_{23} \exists x_{31} \exists x_{32} \exists x_{33} \left(\bigwedge_{i=1}^{3} \bigwedge_{j=1}^{3} x_{ij} \geq 0 \wedge \bigwedge_{k=1}^{3} \left(\sum_{j=1}^{3} x_{kj} = a_k \wedge \sum_{i=1}^{3} x_{ik} = b_k \right) \right)$$

is known to be equivalent to $\sigma \equiv \bigwedge_i (a_i \geq 0) \wedge \bigwedge_j (b_j \geq 0) \wedge \sum_i a_i = \sum_j b_j$. After 0.31 s Asir yields a quantifier-free equivalent ρ containing 84 atomic formulas. So does REDLOG. We automatically check $\forall a_1 \forall a_2 \forall a_3 \forall b_1 \forall b_2 \forall b_3 (\rho \longleftrightarrow \sigma)$ obtaining "true" after 47.83 s plus 36.04 s GC time.

4.3 Kahan's Problem

Write down conditions such that the ellipse $E(x, y) = 0$ with

$$E(x, y) = (x - c)^2/a^2 + (y - d)^2/b^2 - 1$$

is inside the circle $C(x, y) = 0$ with $C(x, y) = x^2 + y^2 - 1$ [Laz88]. We treat the special case $d = 0$. Eliminating

$$\forall a \forall b \Big(b^2(x - c)^2 + a^2 y^2 - a^2 b^2 \neq 0 \vee x^2 + y^2 - 1 \leq 0 \Big)$$

we obtain after 1.21 s plus 0.14 s GC time a quantifier-free formula containing 136 atomic formulas. REDLOG obtains only 59 atomic formulas due degree decreasing techniques, better simplification, and more sophisticated elimination set computation (2.9 s).

4.4 Operation amplifier

This is a problem very close to practice taken from [Hen95]. For the operation amplifier circuit shown in Figure 2, we want to determine the output voltage $\mathrm{V_{OUT}} = \mathrm{v1}$ as a function of the input voltage $\mathrm{V_{IN}} = \mathrm{v\_3}$. We obtain the following algebraic formulation ω of the circuit:

$$-\mathrm{v\_2} + \mathrm{v\_1} + \mathrm{i\_v0} \cdot \mathrm{r1} = 0 \wedge -\mathrm{v\_3} \cdot \mathrm{r1} + \mathrm{v\_2} \cdot \mathrm{r1} + \mathrm{v\_2} \cdot \mathrm{r2} - \mathrm{v\_1} \cdot \mathrm{r2} - \mathrm{i\_pm\_op1} \cdot \mathrm{r1} \cdot \mathrm{r2} = 0 \wedge \mathrm{v\_3} - \mathrm{v\_2} + \mathrm{i\_og\_op1} \cdot \mathrm{r2} = 0 \wedge \mathrm{v\_1} = \mathrm{v1} \wedge \mathrm{v\_3} - \mathrm{v\_og\_op1} = 0 \wedge -\mathrm{v\_pm\_op1} - \mathrm{v\_2} = 0 \wedge \mathrm{vs}^2 \cdot \mathrm{x\_op1}^2 + \mathrm{a} \cdot \mathrm{v\_og\_op1}^2 = \mathrm{a} \cdot \mathrm{vs}^2 \wedge \mathrm{v\_og\_op1} = \mathrm{v\_pm\_op1} \cdot \mathrm{x\_op1}^2 \wedge \mathrm{i\_pm\_op1} = 0$$

The variables to be (existentially) eliminated are

$$\mathrm{V} := \{\mathrm{i\_og\_op1}, \mathrm{v\_2}, \mathrm{i\_pm\_op1}, \mathrm{v\_1}, \mathrm{i\_v0}, \mathrm{v\_pm\_op1}, \mathrm{v\_og\_op1}, \mathrm{x\_op1}\}$$

For the values $\mathrm{a} = 1000$, $\mathrm{vs} = 10$, $\mathrm{r1} = 1000$, and $\mathrm{r2} = 10000$ the translation ω specializes to the following ω':

$$10000 \cdot \mathrm{i\_og\_op1} - \mathrm{v\_2} + \mathrm{v\_3} = 0 \wedge 10000 \cdot \mathrm{i\_pm\_op1} + 10 \cdot \mathrm{v\_1} - 11 \cdot \mathrm{v\_2} + \mathrm{v\_3} = 0 \wedge \mathrm{i\_pm\_op1} = 0 \wedge 1000 \cdot \mathrm{i\_v0} + \mathrm{v\_1} - \mathrm{v\_2} = 0 \wedge \mathrm{v1} - \mathrm{v\_1} = 0 \wedge \mathrm{v\_2} + \mathrm{v\_pm\_op1} = 0 \wedge \mathrm{v\_3} - \mathrm{v\_og\_op1} = 0 \wedge 10 \cdot \mathrm{v\_og\_op1}^2 + \mathrm{x\_op1}^2 - 1000 = 0 \wedge \mathrm{v\_og\_op1} - \mathrm{v\_pm\_op1} \cdot \mathrm{x\_op1}^2 = 0$$

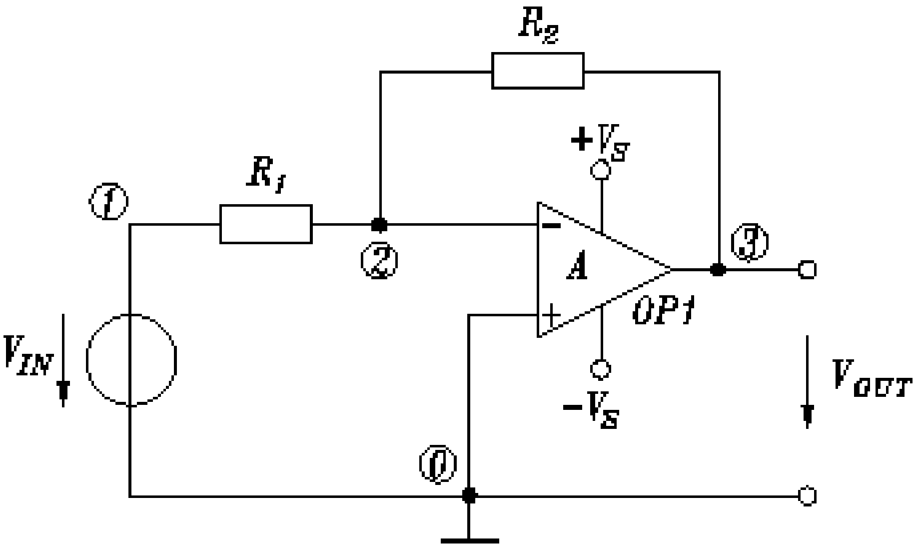

Figure 2: An inverting operation amplifier circuit

A symbolic method usually used for eliminating the variables is the computation of an *elimination ideal basis* wrt. the variables. Using this method, we obtain by using Asir

$$\mathrm{f} := 10 \cdot \mathrm{v\_3}^3 + 100 \cdot \mathrm{v1} \cdot \mathrm{v\_3}^2 - 1011 \cdot \mathrm{v\_3} - 10000 \cdot \mathrm{v1}$$

as implicit description of our function. Figure 3 displays the set of *real* zeroes of f. However, only the part of the curve for $-10 \leq \mathrm{v\_3} \leq 10$ is the correct solution. The other parts are *parasitic solutions* caused by the fact that the elimination ideal is computed over an algebraically closed field, such as $\mathbb{C}$ for our case. In general, it is a problem to distinguish between proper and parasitic solutions.

Since our quantifier elimination is a *real* method, we may expect to get the correct result. The current Asir version is not able to eliminate $\exists V(\omega')$. It would obtain a degree violation when eliminating x_op1 as the final variable. Its degree will be greater than 2 then. We can, however, apply a special case of the degree decreasing methods mentioned in Subsection 3.3 by hand: x_op1 occurs only quadratically. We replace $\mathrm{x\_op1}^2$ by a new variable z in ω' obtaining ω'_z. Similarly we replace x_op1 by z in V yielding V_z. Then we have the equivalence

$$\exists V(\omega') \longleftrightarrow \exists V_z(\omega'_z \wedge z \geq 0),$$

the right hand side of which can be eliminated. The result obtained after 0.03 s contains the correct contraint on the range of v_3 to exclude the parasitic solutions:

$$\underbrace{10 \cdot v\_3^3 + 100 \cdot v1 \cdot v\_3^2 - 1011 \cdot v\_3 - 10000 \cdot v1}_{f} = 0 \wedge \mathrm{v\_3}^2 - 100 \leq 0.$$

Inspection of this example shows that there is actually only Gauss elimination performed, which is a little disappointing. For demonstrating the power of the method, we show how to solve the original parametric problem with REDLOG: For $\exists V(\omega)$, we obtain 604 atomic

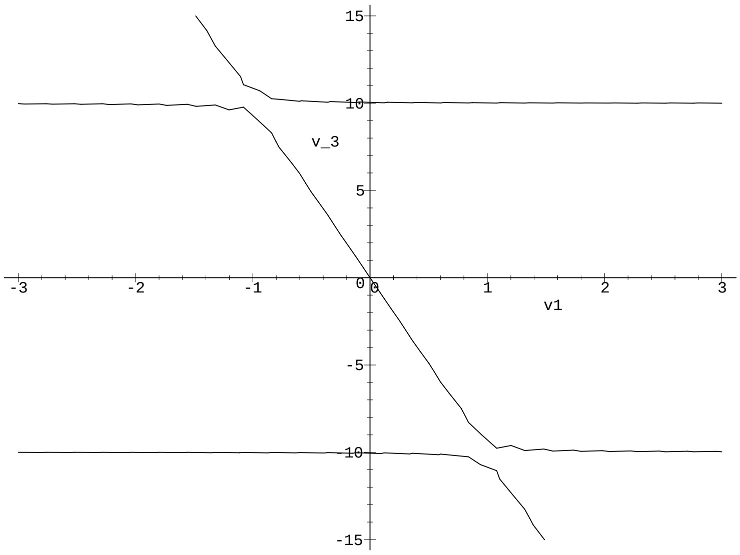

Figure 3: The operation amplifier's behaviour including parasitic solutions for $|\mathrm{v\_3}| > 10$.

formulas after 2.69 s. A result of this size might be useful for further automatic processing but it does not immediately contribute to understanding the network.

We thus apply *generic quantifier elimination* as proposed in Subsection 3. The result obtained for $\exists V(\omega)$ after 0.27 s is:

$$\mathrm{a}\cdot\mathrm{r1}\cdot\mathrm{v\_3}^3-\mathrm{a}\cdot\mathrm{r1}\cdot\mathrm{v\_3}\cdot\mathrm{vs}^2+\mathrm{a}\cdot\mathrm{r2}\cdot\mathrm{v1}\cdot\mathrm{v\_3}^2-\mathrm{a}\cdot\mathrm{r2}\cdot\mathrm{v1}\cdot\mathrm{vs}^2-\mathrm{r1}\cdot\mathrm{v\_3}\cdot\mathrm{vs}^2-\mathrm{r2}\cdot\mathrm{v\_3}\cdot\mathrm{vs}^2= 0 \land \mathrm{a}\cdot\mathrm{v\_3}^2-\mathrm{a}\cdot\mathrm{vs}^2 \leq 0 \land \mathrm{vs} \neq 0,$$

valid under the following conditions:

$$\mathrm{a} \neq 0, \quad \mathrm{r1} \neq 0, \quad \mathrm{r2} \neq 0, \quad \mathrm{v}_3+\mathrm{vs} \neq 0, \quad \mathrm{v}_3-\mathrm{vs} \neq 0, \quad \mathrm{v}_3 \neq 0.$$

None of the conditions is a problem: a is the amplification factor, r1 and r2 are resistors, the absolute value of the output voltage v_3 can certainly never get equal to the supply power vs.

A The Asir user interface

A.1 The formula data type

For the purpose of quantifier-elimination formulas have been introduced to Asir as a new data type. The numerical value corresponding to formulas is 10. Atomic formula operators are collected in Table 3. The First-order operators are displayed in Table 4.

$=$	$\neq$	$\leq$	$<$	$\geq$	$>$
`@==` (`@=`)	`@!=`	`@<=`	`@<`	`@>=`	`@>`

Table 3: Infix operators for the input of atomic formulas (abbreviations for input).

<table>
<tr><th>$\wedge$</th><th>$\vee$</th><th>$\neg$</th><th>$\longrightarrow$</th><th>$\longleftarrow$</th><th>$\longleftrightarrow$</th><th>$\exists x \ldots$</th><th>$\forall x \ldots$</th></tr>
<tr><td>@&&
(@&)</td><td>@||
(@|)</td><td>@!</td><td>@impl</td><td>@repl</td><td>@equiv</td><td>ex(x,...)</td><td>all(x,...)</td></tr>
</table>

Table 4: Binary and infix operators for the input of first-order formulas (abbreviations for input).

We illustrate by example the input of formulas:

```
[1] F = m*x+b@==0 @&& 0@<=x @&& x@<100;
(m*x+b @== 0) @&& (-x @<= 0) @&& (x-100 @< 0)
[2] all(x,F @impl x@<50);
all(x,((m*x+b @== 0) @&& (-x @<= 0) @&& (x-100 @< 0) @impl x-50 @< 0))
[3] ex([x,m],F);
ex(x,ex(m,(m*x+b @== 0) @&& (-x @<= 0) @&& (x-100 @< 0)))
[4] all(@@);
all(b,ex(x,ex(m,(m*x+b @== 0) @&& (-x @<= 0) @&& (x-100 @< 0))))
```

In atomic formulas, all right hand sides are subtracted to the left hand sides at the parsing stage. Notice that the quantifier operators accept also lists of variables. If no variables at all are given, the existential or universal closure of the formula is constructed resp., i.e., all free variables are quantified.

A.2 Functions for formulas

simpl(*f*) A simplified equivalent of *f*. The simplification strategy is described in Subsection 2.8.

qe(*f*) A quantifier-free formula equivalent to *f*. The argument formulas has to obey certain degree restrictions: Quantified variables must not occur with a degree greater than 2. Notice that with the elimination of each quantifier, the degree in the other variables may increase. It thus cannot be determined by inspection of the input whether quantifier elimination will succeed.

atnum(*f*) The number of atomic formulas contained in *f*.

atl(*f*) The set of atomic formulas contained in *f* as a list.

nnf(*f*) A negation normal form of *f*. This is a formula equivalent to *f* which contains only `ex`, `all`, `@&&`, and `@||` as first-order operators.

subf(f,x,t) Substitute all ocurrences of variable x in formula f by polynomial t. This works for quantifier-free formulas. Quantified variables are not yet treated appropriately.

References

[Arn81] D.S. Arnon. *Algorithms for the geometry of semi-algebraic sets.* Ph.d. dissertation, Computer Sciences Department, University of Wisconsin, Madison, 1981. Technical Report No. 436.

[CH91] George E. Collins and Hoon Hong. Partial cylindrical algebraic decomposition for quantifier elimination. *Journal of Symbolic Computation*, 12(3):299–328, September 1991.

[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.

[DH88] James H. Davenport and Joos Heintz. Real Quantifier Elimination is Doubly exponential. *Journal of Symbolic Computation*, 5(1&2):29–35, February 1988.

[DS95a] Andreas Dolzmann and Thomas Sturm. *Redlog, a Reduce Logic Package.* FMI, Universität Passau, D-94030 Passau, Germany, preliminary edition, July 1995. User Manual.

[DS95b] 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.

[DS96] Andreas Dolzmann and Thomas Sturm. Redlog—computer algebra meets computer logic. Technical Report MIP-9603, FMI, Universität Passau, D-94030 Passau, Germany, February 1996.

[DSW96] Andreas Dolzmann, Thomas Sturm, and Volker Weispfenning. A new approach for automatic theorem proving in real geometry. Technical Report MIP-9611, FMI, Universität Passau, D-94030 Passau, Germany, May 1996.

[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.

[Hen95] Eckhard Henning. Rechnergestützte Dimensionierung analoger Schaltungen auf der Basis symbolischer Analyseverfahren. Technical report, Zentrum für Mikroelektronik, December 1995. Jahresbericht zum Forschungsprojekt.

[Laz88] Daniel Lazard. Quantifier elimination: Optimal solution for two classical examples. *Journal of Symbolic Computation*, 5(1&2):261–266, February 1988.

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

[NT92] M. Noro and T. Takeshima. Risa/Asir—a computer algebra system. In *Proceedings of the ISSAC '92*, pages 387–396, 1992.

[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.

[Wei94a] Volker Weispfenning. Parametric linear and quadratic optimization by elimination. Technical Report MIP-9404, FMI, Universität Passau, D-94030 Passau, Germany, April 1994. To appear in the Journal of Symbolic Computation.

[Wei94b] 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.

[Wei96a] Volker Weispfenning. Applying quantifier elimination to problems in simulation and optimization. Technical Report MIP-9607, FMI, Universität Passau, D-94030 Passau, Germany, April 1996. To appear in the Journal of Symbolic Computation.

[Wei96b] Volker Weispfenning. Quantifier elimination for real algebra—the quadratic case and beyond. To appear in AAECC, 1996.